Community Identification

Suyash Parth (05CS1024)

April 4, 2009

1 Introduction

In the simplest case, a network or graph can be represented as a set of points or vertices, joined in pairs by lines or edges. Many networks, it is found, are inhomogeneous consisting not of an undifferentiated mass of vertices, but of different groups.Within these groups there are many edges between vertices, but between groups there are fewer edges. These groups are called communities.

Our task is to identify these communities in the network structure.The ability to find communities in some automated fashion could be of considerable use. For example, communities in a web graph might correspond to sets of web sites dealing with related topics. So, taking into consideration the assumption that one vertex belongs to one community only and the network is undirected, we discuss here some algorithms for the extraction of communities from raw network data.

Here, we describe some of the historical approaches to find communities including spectral bisection method and hierarchial clustering method. As well as, we describe some newer methods that have appeared in the last few years, including the edge betweenness method of Girvan and Newman and a number of variations proposed by other authors.

The problem of identifying communities in networks is tackled in two ways. First, by partitioning the graph(the network represented as a graph) without removing any edge and second, by getting a hierarchy of communities by subsequent edge removals.

There has been considerable recent interest in algorithms for finding communities in networks groups of vertices within which connections are dense, but between which connections are sparser. Here we review the progress that has been made towards this end. We begin by describing some traditional methods of community detection, such as spectral bisection, the Kernighan-Lin algorithm and hierarchical clustering based on similarity measures. None of these methods, however, is ideal for the types of real-world network data with which current research is concerned, such as Internet and web data and biological and social networks. We describe a number of more recent algorithms that appear to work well with these data, including algorithms based on edge betweenness scores, on counts of short loops in networks and on voltage di®erences in resistor networks.

2 Definitions of community

2.1 Strong Definition

Given that there are two groups G_1 and G_2.
Let $h \in G_1, i \in G_2, j \in G_2$.
We define : k_{ih} as the number of edges from i to h (also written as i $\leftrightarrow$ h).
If

$$k_{ij} > k_{ih} \forall i \in G_1 \tag{1}$$

then G_1 is a community.

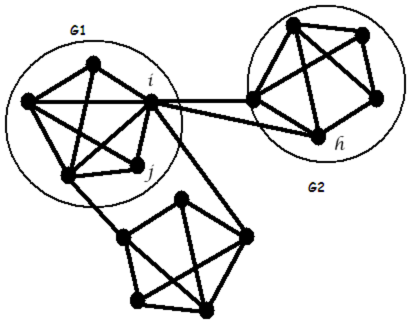

Figure 1: Example of a strong community

In Fig(1), for every member of the group(G_1), the number of edges from it inside the group is greater than number of edges outside the group, hence (G_1) follows the strong definition of community.

2.2 Weak Definition

Given that there are two groups G_1 and G_2.
Let $h \in G_1, i \in G_1, j \in G_2$.

We define : k_{ih} as the number of edges from i to h (also written as i $\leftrightarrow$ h).
If If

$$\sum_{\forall i \in G_1} k_{ih} > \sum_{\forall i \in G_1} k_{ij} \tag{2}$$

then G_1 is a community.

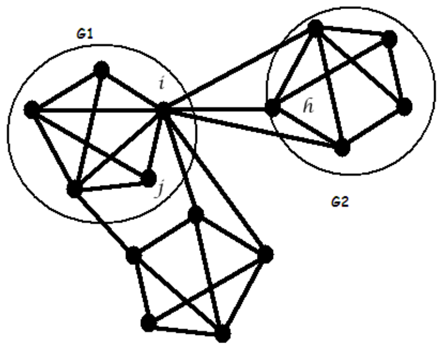

Figure 2: Example of a week community

In Fig(2), the member i of the group(G_1) has five edges outside the group and only four inside the group, therefore G_1 doesnot follow the strong definition of community. However, as the sum of the edegs from G_1 outside it ($= 6$) is less than the number of edges inside G_1 ($= 9$), hence G_1 follows the week definition of community.

2.3 Hierarchical Definition

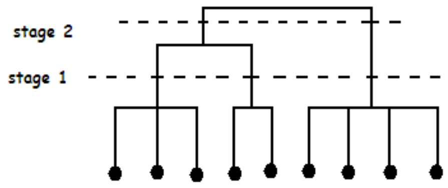

Figure 3: The dendogram shows hierrarchical aglomerative clustering. We start with each node as a seperate community and communities merge to generate bigger communities.

Initially all the nodes are assumed in different communities. This means that there exists n number of communities, where n is the number of vertices. After that using **Hierarchical Clustering or Structural equivalence** different communities are joined together. This is done by using some **threshold**. At the final stage, we end up with some communities which can not be merged. Alternatively, we may start with all the nodes in a single cluster and use edge-betweenness(4.1) or some similar parameters to sub-divide the community into more closle related communities. Thus, communities are decided hierarchically.

3 Graph Partitioning Algorithms

3.1 The Spectral Bisection Method

The **Laplacian** of an n-vertex undirected graph G is a $n \times n$ symmetric matrix **L** whose diagonal element L_{ii} is the degree of vertex i and whose off-diagonal element L_{ij} is -1 if vertices i and j are connected by an edge and zero otherwise.

Assuming that d_i is the degree of node i then Laplacian matrix is defined as:

$$L_{ij} = \begin{cases} d_i & \text{if } i = j; \\ -1 & \text{if } i \text{ is connected to } j; \\ 0 & \text{Otherwise.} \end{cases} \tag{3}$$

We can also write **L = D - A** , where **D** is the diagonal matrix of vertex degrees and **A** is the adjacency matrix.

The eigenvalues and eigenvectors of the laplacian matrix(L) are defined by the following equation:

$$\mathbf{LX} = \lambda \mathbf{IX} \tag{4}$$

where λ = Eigenvalue, X = Eigenvector and I = Identity matrix.

Since the degree,

$$d_i = D_{ii} = \sum_j^n A_{ij} = \sum_i^n A_{ij} \tag{5}$$

it follows that all rows and columns of the Laplacian sum upto zero, and hence the vector **1** (all 1's), is always an eigenvector for the matrix **L** with eigenvalue zero ($M_i \ = \sum_j^n L_{ij} \times 1 = d_i$ - $\sum_j^n A_{ij} = 0$).

Since, all eigenvectors corresponding to non-degenerate eigenvalues for a real symmetric matrix are orthogonal, all eigenvectors other than that corresponding to the lowest eigenvalue must have both positive and negative elements(6.1). So, for the case of two **weakly coupled communites** there will be one eigenvector with eigenvalue greater than zero (may be called second lowest eigenvalue) and elements all positive for one community and all negative for the other. This is because, all elements are nearly equal within a community. Thus, a network can be divided into its two communities by looking at the eigenvector corresponding to the second lowest eigenvalue.

Let us consider the following example:

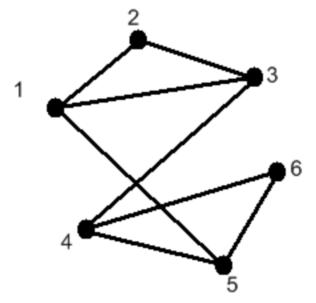

Figure 4: Network for showing spectral bisection method

We shall try to partition the above network into 2 communities using the eigenvectors(spectral bipartitioning). We compute the Laplacian matrix(L) corresponding to this network to be:

$$L = \begin{pmatrix} 3 & -1 & -1 & 0 & -1 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & 0 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{pmatrix} \tag{6}$$

Solving eq(4), we get

$$det(L - \lambda I) = 0; \tag{7}$$

Substituting the value of L from eq(6) into eq(7) we get the following characterstic equation in λ:

$$\lambda^6 - 16\lambda^5 + 98\lambda^4 - 284\lambda^3 + 381\lambda^2 - 180\lambda = 0 \tag{8}$$

Therefore,

$$\lambda = \begin{pmatrix} 0 & 1 & 3 & 3 & 4 & 5 \end{pmatrix} \tag{9}$$

Here 1 is the 2nd lowest eigen value, the eigenvector corresponding to this gives the bipartition. Substituting λ in eq(4), we get the corrosponding eigenvector to be

$$X = \begin{pmatrix} -1 \\ -2 \\ -1 \\ 1 \\ 1 \\ 2 \end{pmatrix} \tag{10}$$

So,nodes 1, 2, 3 are in one community and nodes 4, 5 and 6 are in the other community. We observe that the network partitions well but not perfectly into communities,

$$L = \begin{pmatrix} & & & 0 & -1 & 0 \\ & B_1 & & 0 & 0 & 0 \\ & & & -1 & 0 & 0 \\ 0 & 0 & -1 & & & \\ -1 & 0 & 0 & & B_2 & \\ 0 & 0 & 0 & & & \end{pmatrix}$$

there are few edges that do not fit the block-diagonal pattern(as shown).

3.2 Kernighan-Lin Algorithm

The main aim of **Kernighan-LIn Algorithm** is to decompose a complex system into smaller subsystems so that every subsystem may be designed independently. Decomposition scheme has to minimize the interconnections among the subsystems. This decomposition is carried out hierarchically until each subsystem is of managable size. This algorithm treats nodes as chips on a circuit board and edges as interconnections between chips. Circuit is partitioned into parts in such a way that every component is within a prescribed range and the number of connections among the components is minimized.

This algorithm follows **iterative 2-way balanced partitioning** heuristic.

The algorithm begins by dividing the graph into 2 random groups.
We define:
$E_1 \rightarrow$ sum of 'across-board connections' from chip v_1 to the other group
$I_1 \rightarrow$ sum of 'intra-board connections' from v_1 to other chips in its group
$\gamma_{24} \rightarrow$ the weight of the connection between v_2 and v_4
$\Delta_{ij} \rightarrow$ the Gain by swapping groups of v_i and v_j
$D_i = E_i - I_i$
$\Delta_{ij} = D_i + D_j - \gamma_{ij}$

The **Algorithm** is as follows:

1. Randomly divide the nodes into two groups of size n_1 and n_2.
2. Initialize i with 1.
3. Iterate over i:
4. Initialize j with 0.
5. Increase j by 1.
6. Take all possible pairs between the groups.
7. Calculate benefit function Δ_{ij} for all the pairs if swapped.
8. Exchange the vertex pairs which have the maximum benefit function(*this pair when swapped results in the largest decrease or smallest increase in cut size of the graph*).
9. These vertices are then locked and thus are prohibited from participating in any further exchanges.
10. Repeat step 5 through 10 until all the vertices of atleast one group are locked.
11. Find the set with the largest partial sum for swapping. This largest partial sum is stored as Q_i. It is defined as $\sum_{j=1}^{k} \Delta_{ij}$ where k is varies from **1 to** n to get the maximum sum.
12. If Q_i is greater than zero, swap the nodes numbered from 1 to k . Here k is the index, where maximum partial sum was evaluated.
13. If Q_i is less than or equal to zero, do nothing and come out of the algorithm.
14. Unlock all vertices.
15. Increase i by 1.
16. Repeat the steps 3 through 16.

Since, this algorithm depends upon initial partition, we have no idea apriori what the sizes of the group initially will be, hence it will be unsuitable for most applications of real world network. Besides, where as spectral bisection method has worst-case running time of $O(n^2)$, this method in worst case takes $O(n^4)$ time. But, if the initial size of groups may be chosen optimally, then this method will take $O(n^3)$ time in worst case and give us the perfect partitioning.

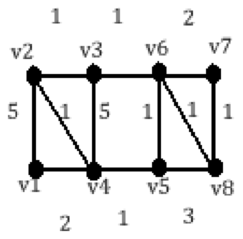

Figure 5: Example network to show Kernighan-Lin Algorithm

Let us workout an example (Fig:5).
Let the initial sets be

$$S_1 = \begin{pmatrix} v_1 & v_4 & v_5 & v_8 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} v_2 & v_3 & v_6 & v_7 \end{pmatrix}$$

$$D_i = \begin{pmatrix} v_1 & v_4 & v_5 & v_8 & & v_2 & v_3 & v_6 & v_7 \\ 3 & 3 & -3 & -1 & & 5 & 3 & -1 & -1 \end{pmatrix}$$

Swapping v_2 and v_4 provide the highest benifit(Δ_{24} = 6). Hence, we then lock v_2 and v_4 and look at the remaining nodes.

$$D_i = \begin{pmatrix} v_1 & v_4 & v_5 & v_8 & & v_2 & v_3 & v_6 & v_7 \\ -3 & & -1 & -1 & & & -5 & -1 & -1 \end{pmatrix}$$

Now swapping v_7 and v_5 gives the least loss(Δ_{57} = -2). We now lock v_5 with v_7 and lock further.

$$D_i = \begin{pmatrix} v_1 & v_4 & v_5 & v_8 & & v_2 & v_3 & v_6 & v_7 \\ -3 & & & 3 & & & -5 & -1 & \end{pmatrix}$$

Lock v_6 and v_8. $\Delta 68 = 2$.

$$D_i = \begin{pmatrix} v_1 & v_4 & v_5 & v_8 & & v_2 & v_3 & v_6 & v_7 \\ -3 & & & & & & -3 & & \end{pmatrix}$$

Δ_{13} = -6.

We now see the largest sequence for which there is gain. Hence by swapping $\begin{pmatrix} v_2 & v_7 & v_6 \end{pmatrix} \begin{pmatrix} v_4 & v_5 & v_8 \end{pmatrix}$ in this iteration for a gain six. We continue the process till there is some gain and then terminate.

3.3 Wu Huberman algorithm

This algorithm considers the network as a circuit. In this case, two nodes of the network are attached to the two ends of 1 Volt battery.1 Ohm resistance is inserted in all other edges. A threshold is considered. If a particular node is above the threshold, then it is in one community , otherwise it is in other community.

From the **Kirchoff's Current Law**, we know that the net current at ith node is NIL. So, we get the following for the i-th node:

$$\sum_j I_{ij} = 0;$$

$$\sum_j \frac{V_i - V_j}{R} = 0;$$

$$\sum_j (V_i - V_j) = 0, since, R = 1.$$

$$V_i = \frac{1}{k_i} \sum_j V_j;$$

Here, k_i is the number of neighbors of i-th node.

$$V_i = \frac{1}{k_i} \sum_{jadjacenttoi} V_j;$$

$$V_i = \frac{1}{k_i} \sum_{V(G)} V_j a_{ij};$$

$$V_i = \frac{1}{k_i} \sum_{V(G)-\{i,1\}} V_j a_{ij} + \frac{1}{k_i} a_{i1};$$

This equation leads to three matrices.

$$V = \begin{pmatrix} V_3 \\ V_4 \\ . \\ . \\ . \\ V_n \end{pmatrix}$$

$$B = \begin{pmatrix} \frac{a_{33}}{k_3} & \frac{a_{34}}{k_3} & \frac{a_{3n}}{k_3} \\ \frac{a_{43}}{k_4} & \frac{a_{44}}{k_4} & \frac{a_{4n}}{k_4} \\ \frac{a_{53}}{k_5} & \frac{a_{54}}{k_5} & \frac{a_{5n}}{k_5} \\ \frac{a_{n3}}{k_n} & \frac{a_{n4}}{k_n} & \frac{a_{nn}}{k_n} \end{pmatrix}$$

$$C = \begin{pmatrix} \frac{a_{31}}{k_3} \\ \frac{a_{41}}{k_4} \\ . \\ . \\ . \\ \frac{a_{n1}}{k_n} \end{pmatrix}$$

So, Now, $V = BV + C$; $V = (I - B)^{-1}C$;

$$V = \sum_{m=0}^{+\infty} B^m C;$$

Using this equation,we can get the voltages of all the nodes. Generally, 0.5 V is assumed threshold. So, the nodes, which have voltage less than 0.5V falls in one group and the others come in other group.

We can get the voltages of all the nodes by a second method,which is nothing but iteration.We define a matrix of initial voltages V_I.Now, using the formula,used in first method, we can get the final matrix of voltages in subsequent iterations.An iteration is assumed final when $f^{r-1}(v)$ and $f^r(v)$ are equal. Let

$$f^1(v) = BV_I + C;$$

$$f^2(v) = B(BV_I + C) + C;$$

$$f^r(v) = B^r V_I + \sum_{i=0}^{r-1} B^i C;$$

if $V_I = 0$

$$f^r(v) = \sum_{i=0}^{r-1} B^i C;$$

Thus the network can be partitioned. The series will converge. The speed of convergence is given by - $||B||$.

Each equation requires k_i operations. Thus, the complexity of the algorithm $= \sum k_i = o(e)$.

Let us now revisit the old example:

Since,the voltage source is of **1 V** and resistors are of resistance **1 Ohm**, Kirchoff's law can be applied easily. We get:

$V_6 = 1V$
$V_4 = V_5 = 0.66V$
$V_1 = V_3 = 0.33V$
$V_2 = 0V$

Threshold Voltage(V_{th}) $= 0.5V$

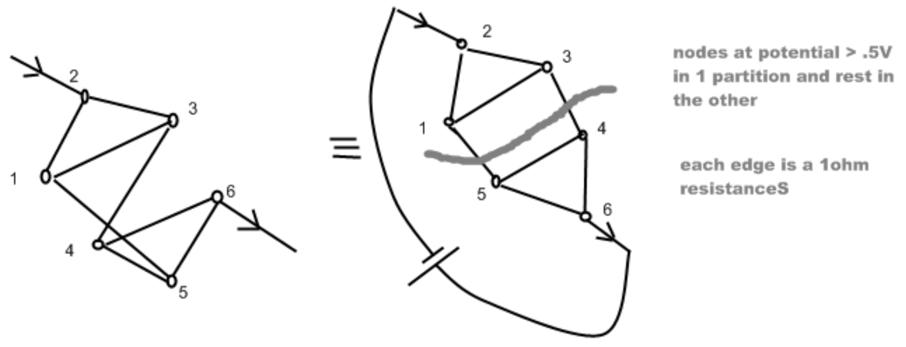
1
2
3
4
5
6
≡
1
2
3
4
5
6
nodes at potential > .5V in 1 partition and rest in the other
each edge is a 1ohm resistanceS

Figure 6: applying Wu-huberman across nodes 2 and 6

So, nodes 1, 2 and 3 are in one group, whereas nodes 4, 5 and 6 are in the other.

4 Methods based on edge removal(divisive approach)

All the methods mentioned hereon ask a common question: *whether an edge is a community edge or not.* We try removing the non-community edges so as to get our partitions(communities).

4.1 The algorithm of Girvan and Newman

This algorithm follows the **divisive** method in contrast with the **agglomerative hierarchial clustering** method, since , here edges are removed from a network.

This algorithm assumes the edges between the communities as "bottlenecks" because any traffic from one community to another will flow through this. So,we look for the edges between the communities with highest traffic.Removing these should split the network into its natural communities.

Initially the entire graph is assumed as a cluster.Then,

1. Calculate edge betweenness of each node.
2. Remove the edge with highest edge betweenness.
3. Recalculate the edge betweenness again.

Edge betweeness is defined as any one of them

1. Number of shortest paths through a edge.
2. Amount of current passing through that edge.
3. Expected number of random walkers passing through that edge.

$$E = \begin{pmatrix} e_{11} & e_{12} & e_{1m} \\ e_{21} & e_{22} & e_{2m} \\ e_{31} & e_{32} & e_{3m} \\ e_{m1} & e_{m2} & e_{mm} \end{pmatrix}$$

Let there are m clusters.

Here e_{ij} is the number of edges between cluster i and cluster j and the diagonal element e_{ii} represents the number of edges between cluster i only. Ideally, non-diagonal elements should be zero(since in ideal cluster the components should be disconnected from each other). So, deviation from this behavior can be a co-efficient for randomness of the network. We define the co-efficient as follows,

$$Q = \sum_{\forall i} (e_{ii} - a_i^2)^2$$

where,

$$a_i = \sum_{j=1}^{k} e_{ij}.$$

Also, the term $(e_{ii} - a_i^2)$ is denoted as the **affinity** of the i-th cluster.

In other words, Q is the fraction of all edges that lie within communities minus the expected value of the same quantity in a graph in which the vertices have the same degrees but edges are placed at random without regard for the communities.For example, a value of $Q = 0$ indicates that the community structure is no stronger than would be expected by random chance and values other that zero represent deviations from randomness.

The worst case time complexity of this algorithm is $O(n^3)$ on a sparse graph, because there are n edges to be removed in total and each iteration of the algorithm takes $O(n^2)$.This time complextiy,being slow,is a disadvantage for this algorithm.

4.2 Shortest Path Edge Betweenness

Betweenness of an edge is defined to be the number of shortest paths between vertex pairs that run along the edge in quetion, summed over all vertex pairs. This quality can be calculated for all edges in time that goes as **O(mn)** on a graph with m edges and n vertices.

Assume that d is depth and w is the number of paths passing through the node.The algorithm works in two phases. First,we apply top down approach and then bottom up approach. In top down approach, we calculate the depth d and weight w for all the nodes. In Bottom up approach, Edge betweenness of all the edges are calculated.

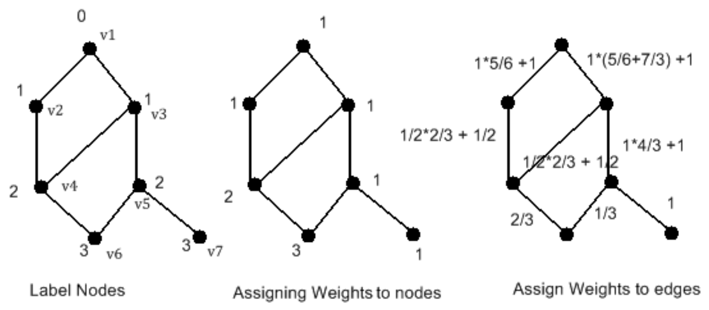

Figure 7: step by step computation of shortest path betweenenss

Top Down Approach

1. Initialize vertex(s) $d_s = 0, w_s = 1$
2. After visiting node i mark that node.Then goto node j
3. If node j is not marked
   $d_j = d_i + 1; w_j = w_i;$
4. If node j is marked and if $d_j = d_i + 1$
   $w_j = w_j + w_i$;
5. If node j is marked and if $d_j < d_i + 1$
   Do nothing

Bottom Up Approach

1. Start from the node i which has maximum depth.

2. Edge Betweenness **EB**=$\frac{w_j}{w_i}$ where j is the neighbor of i

3. For other nodes j close to source,this means that $d_j < d_i$
   EB= (incoming EB +1)*$\frac{w_j}{w_i}$, where j is the neighbor of i

Edge Betweenness **EB** of network is defined as the average of edge betweenness of all the edges. This means that

$$EB = \frac{\sum_{i=1}\sum_{j=1} EB_{ij}a_{ij}}{\sum_{i=1}\sum_{j=1} a_{ij}}$$

where a_{ij} is an entry in adjacency matrix.

For the example shown, we first label the depth of each node with refrence to the starting node. We then compute the weight of the nodes which is the sum of their parent nodes. We should not that these weights are not arbitrary values, they are the number of shortest paths from the source to the particular node. Now, we go bottom-up and assign weights to edges. At any level i, the sum of the edge weights from a node at level i to a node(n_k) at $i + 1$ is proportional to the number of shortest paths from the source, that will pass through that node(n_k). We distribute this weight proportionally among the parent nodes of n_k(the first expression in edge weights *eg* 2/3 for v_4). We also add the paths from parents to the node(*eg* 1/2 for v_4).

4.3 Random Walker on the bridge edge(Finding effective movement)

Suppose that signals do not travel along geodesic paths, but instead just perform a random walk about the network until they reach their destination. This gives us another measure on edges, the random-walk betweenness: we calculate the expected net number of times a random walker walking between a particular pair of vertices will passes down a particular edge and sum over all vertex pairs. The random-walk betweenness can be calculated using matrix methods.

Effective Movement is defined as the difference of number of times going and number of times coming back.

Let i, j be two nodes with degrees k_i and k_j respectively. So, the probability of an agent going from j to $i = \frac{A_{ij}}{k_j}$. Here A_{ij} is an entry in **adjacency matrix A**.

We define:

$$D^{-1} = \begin{pmatrix} \frac{1}{k_1} & \frac{0}{k_2} & . & . & . & \frac{0}{k_n} \\ \frac{0}{k_1} & \frac{1}{k_2} & . & . & . & \frac{0}{k_n} \\ . & . & . & & & \\ . & . & . & & & \\ \frac{0}{k_1} & \frac{0}{k_2} & . & . & . & \frac{0}{k_n} \\ \frac{0}{k_1} & \frac{0}{k_2} & . & . & . & \frac{1}{k_n} \end{pmatrix}$$

$M = AD^{-1}$ = probability of reaching from i to j in one step.

$$M = \begin{pmatrix} \frac{a_{11}}{k_1} & \frac{a_{12}}{k_2} & . & . & . & \frac{a_{1n}}{k_n} \\ \frac{a_{21}}{k_1} & \frac{a_{22}}{k_2} & . & . & . & \frac{a_{2n}}{k_n} \\ . & . & . & & & \\ . & . & . & & & \\ \frac{a_{31}}{k_1} & \frac{a_{32}}{k_2} & . & . & . & \frac{a_{3n}}{k_n} \\ \frac{a_{n1}}{k_1} & \frac{a_{n2}}{k_2} & . & . & . & \frac{a_{nn}}{k_n} \end{pmatrix}$$

M^n = probability of reaching from i to j after n steps. However, this M^n includes paths in which the destination j is reached more than once. We need to compute the effective movement for our purpose, hence we need to modify M slightly. Assuing s as start node and t as target node. We define $M_t = A_t D_t^{-1}$. Here A_t is the reduced adjcency matrix without target node. So, $(M_t^n)_{is}$ shows the matrix of transition probabilities from s to i in n steps where we haven't yet reached the destination t.

Total Probability of reaching from s to u=

$$\frac{1}{k_u}(M_t^0 + M_t^1 + M_t^2 + ... + M_t^\infty);$$

$$P_{us} = \frac{1}{k_u}[(I - M_t)^{-1}]_{us};$$

So, we get the probability of moving from u to v as

$$P_{u \to v} = |P_{us} - P_{vs}|$$

For all edges uv, the random-walk betweeness for the source s and destination t can be shown to be:

$$P = D^{-1}(I - M_t)^{-1}S;$$

where, the source vector s is the vector whose components are all 0 except for a single 1 in the position corresponding to the source vertex s.

This algorithm takes $o((m + n)mn^2)$ time, where m is the number of edges and n is the number of nodes.

4.4 Radicchi's Algorithm

This algorithm , like that of Girvan and Newman , is based on iterative removal of edges, but uses a different measure instead of betweenness to identify the edges to be removed. As in the algorithm of Girvan and Newman, this measure is calculated after each removal, but it is a local measure that can be calculated quickly and hence the overall algorithm runs faster.

The algorithm is based on counting the short loops of edges in the network - loops of length three, or trianges, in the simplest case. Here, one should be able to spot the between-community edges by looking for ones that belong to unusually small number of loops.

Radicchi's algorithm tries to find the **edge-clustering coeffcient** of each edge. Depending on that coefficient, we remove the edge with the smallest value of the coefficient from the network. It finds the number of triangles a particular edge is part of.

Let, n_1 and n_2 be two nodes which are in two different clusters.

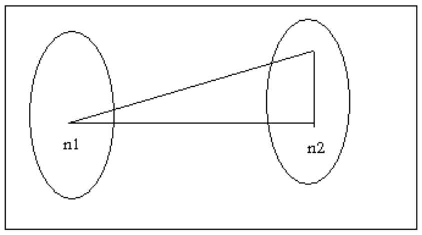

Figure 8: Triangle formation by nodes n_1 and n_2

Here , **Edge-clustering coefficient** c_{ij} is defined as

$$c_{ij} = \frac{Z_{ij} + 1}{\min(k_i - 1, k_j - 1)};$$

Z_{ij} = Number of triangles passing through i and j.
$\min(k_i - 1, k_j - 1)$ = Minimum excess degree = Maximum number of triangles possible.
We remove the edge with the lowest value of c_{ij}, which is actually acting like a bridge between the two clusters.

This algorithm runs in time $O(m^4/n^2)$ on a graph with m edges and n vertices or $O(n^2)$ on a sparse graph, which in one order of system size faster than Newman-Girvan algorithm. It is a fast technique and hence, most widely used.

Let us work the following example (Fig(9)):

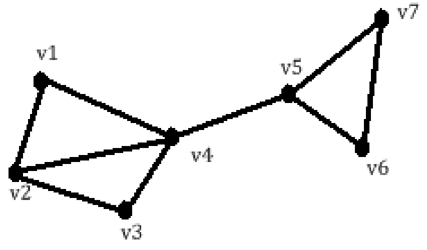

Figure 9: Example for radicchi's

The community coefficient for the edges are :

$$c = \begin{array}{ccccccccc} e_{12} & e_{14} & e_{23} & e_{24} & e_{34} & e_{45} & e_{56} & e_{57} & e_{67} \\ 2 & 2 & 2 & 1.5 & 2 & 0.5 & 2 & 2 & 2 \\ 2 & 2 & 2 & 1.5 & 2 & & 2 & 2 & 2 \\ 1 & 1 & 1 & & 1 & & 2 & 2 & 2 \end{array}$$

We remove edge e_{45} after the first iteration which breaks the network into two communities (v_1, v_2, v_3, v_4) and (v_5, v_6, v_7). In the next iteration removing e_{23} doen't give us anymore communities and all the edges get same value so we stop.

4.5 Radicchi's Algorithm for Weighted Networks

Edges that run between communities are unlikely to belong to many short loops, because to complete a loop containing such an edge there needs to be another edge that runs between the same two communities and such other edges are rare.

So, the **Radicchi's Algorithm** is modified for weighted network. For, weighted networks, rather than considering simply the triangles, we need to consider the weights on the edges forming these triangles. The basic idea is that,if the weights on the edges forming a triangle are comparable then there is a high chance of being those in the same community. On the other hand, if those are not comparable, then they will be in different communities.

To capture this property, we define a strength matrix **S** for each of the edges in the network.

Let the weight of the $edge(u, v)$, where $u, v \in V$, be denoted by w_{uv} .
We define S as,

$$S = \frac{w_{uv}}{(\sum_{i \in V-u,v}(w_{ui} - w_{vi})^2)^{\frac{1}{2}}};$$

Network may be partitioned into clusters or communities by removing edges that have **S** close to zero.

5 Conclusion

Here, we have reviewed algorithmic methods for identifying communities of densely connected vertices in a network. We have discussed some of the traditional approaches, such as, spectral graph partitioning and hierarchical clustering. But, as we have pointed out, these have a number of shortcomings as far as the analysis of large real-world networks is concerned. In more recent times several new methods have been developed that are fast and flexible enough to apply to quite general network structures. We have described several methods based on iterative removal of edges based on their community-betweenness proposed by Girvan and Newman, as well as the algorithm based on counts of short loops proposed by Radicchi. We have had a look at the time complexities of these algorithms and outlined their strengths and weaknesses. As a result of substantial progress in recent years, it appears we now have an effective toolkit for studying community structure in networks. There is certainly still room for improvement however in both speed and sensitivity of community structure identification algorithms, and there are many interesting networked systems awaiting analysis using these methods.

6 Appendix

6.1 Perron-Frobenius Theorem

A is a nonnegative irreducible nxn matrix if and only if:

$$(I_n + A)^{n-1} > 0$$

where, n as in the size of the matrix.

Theorem 1 : If A is nxn, nonnegative, irreducible, then:

1. One of its eigenvalues is positive and greater than or equal to (in absolute value) all other eigenvalues.
2. There is a positive eigenvector corresponding to that eigenvalue, every other eigenvector for smaller eigenvalues has negative components.
3. And, that eigenvalue is a simple root of the characteristic equation of A.

Such an eigenvalue is called the "dominant eigenvalue" of A and we assume hereafter we have numbered the eigenvalues so it is the first one. We should point out that other eigenvalues may be positive, negative or complex (and if they are complex then by "absolute value" we mean modulus, or distance in the complex plane from the origin). Symmetric matrices are guaranteed to not have complex eigenvalues. Hence any undirected graph will have real eigenvalues(since L is symmertic).

6.2 Comparative Study

Algorithm	Type	Complexity
Spectral bisection	Partitional	O(n3)
Kernighan-Lin	Partitional	O(n2)
Wu and Huberman	Partitional	O(n+m)
Girvan and Newman	Hierarchical Divisive	O(m2n) or O(n3)
Radicchi	Hierarchical Divisive	O(m4/n2) or O(n2)