1.

1 MIT: Community Detection and the Stochastic Block Model

1.1 Community Detection

The problem of community detection, is that of minimum graph bisections. We want to bisect a graph into (S, S^C), where $|S| = |S^C|$ and $S \cap S^C =$, while minimizing cut(S)

1.2 Stochastic Block Model

Consider a random graph where $n \in \mathbb{Z}^+$ is an even integer. Then let A, B be sets of nodes such that $|A| = |B| = \frac{n}{2} = m$. We pose the following question: for each pair of (i, j) nodes, the edge (i, j) comes with a probability p that if i and j are in the same set (either A or B), and probability $q = 1 - p$ that they are not in the same set. [Read more on random graphs] This is the **Stochastic Block Model** on two communities A and B.

We will want to recover the original partition. The problem is incredibly difficult if $p = q$ (50/50), but trivial when $p = 1$ and $q = 0$ (perfect knowledge). So we'll at what values of p and q can this recovery be done?

Assume $p > q$, and we'll attempt to find the original partition by attempting to find a minimum bisection of the graph given.

1.3 What Does the Spike Model Suggest?

We'll use a method of spectral clustering to partition the graph.

Let A be the adjacency matrix of G :

$$A_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E(G) \\ 0 & \text{otherwise.} \end{cases}$$

Note that A is a random matrix, since G is a random graph. We would like to solve the

problem:

$$\begin{aligned} \max & \sum_{i,j} A_{i,j} x_i x_j \\ \text{s.t. } & x_i = \pm 1, \forall i \\ & \sum_j x_j = 0. \end{aligned}$$

The solution x will take on $+1$ in one cluster and -1 in the other.

[Equivalently this is:

$$\begin{aligned} & \max \langle Ax, x \rangle \\ s.t. & x_i = \pm 1, \forall i \\ & \sum_j x_j = 0 \end{aligned}$$

]

Relaxing the condition of $x_i = \pm 1$ for all i to $\|x\|_2^2 = n$ would yield a spectral method:

$$\begin{aligned} \max & \sum_{i,j} A_{i,j} x_i x_j \\ \text{s.t.} & \|x\|_2 = \sqrt{n} \\ & \mathbf{1}^T x = 0 \end{aligned}$$

The solution of this problem consists of taking the top eigenvector of the projection of A on the orthogonal of the all-ones vector $\mathbf{1}$.

The matrix A is a random matrix with an expectation matrix given by:

$$\mathbf{E}[A] = \begin{cases} p & \text{if } (i,j) \in E(G) \\ q & \text{otherwise} \end{cases}.$$

Let g denote a vector that is $+1$ on the clusters and -1 in the other (this is what we want to recover). Then we can write:

$$\mathbf{E}[A] = \frac{p+q}{2} \mathbf{1}\mathbf{1}^T + \frac{p-q}{2} g g^T,$$

and

$$A = (A - \mathbf{E}[A]) + \frac{p+q}{2} \mathbf{1}\mathbf{1}^T + \frac{p-q}{2} g g^T$$

We want to remove the term $\frac{p+q}{2}\mathbf{1}\mathbf{1}^T$ we consider the random matrix:

$$\mathcal{A} = A - \frac{p+q}{2}\mathbf{1}\mathbf{1}^T$$

Applying $\mathbf{E}$ to $\mathcal{A}$:

$$\begin{aligned}\mathbf{E}[\mathcal{A}] &= \mathbf{E}[A - \frac{p+q}{2}\mathbf{1}\mathbf{1}^T] \\ &= \mathbf{E}[A] - \mathbf{E}[\frac{p+q}{2}\mathbf{1}\mathbf{1}^T] \\ &= \mathbf{E}[A] - \frac{p+q}{2}\mathbf{E}[\mathbf{1}\mathbf{1}^T]\end{aligned}$$

It's easy to see that:

$$\mathcal{A} = (\mathcal{A} - \mathbf{E}[\mathcal{A}]) + \frac{p-q}{2}gg^T.$$

That is $\mathcal{A}$ is a superposition of a random matrix whose expected value is 0, and a rank-1 matrix; i.e.:

$$\mathcal{A} = W - \lambda vv^T$$

where $W = (\mathcal{A} - \mathbf{E}[\mathcal{A}])$ and $\lambda vv^T = \frac{p-q}{2}n\left(\frac{g}{\sqrt{n}}\right)\left(\frac{g}{\sqrt{n}}\right)^T$. For large λ, we showed that the eigenvalue associated with λ pops outside the distribution of eigenvalues of W and whenever this happens, the leading eigenvector has a non-trivial correlation with g (the eigenvector associated with λ).

Note that since to obtain $\mathcal{A}$ we simply subtract a multiple of $\mathbf{1}\mathbf{1}^T$ from $\mathcal{A}$, the maximization problem is equivalent to:

$$\begin{aligned}\max \sum_{i,j} \mathcal{A}_{i,j}x_ix_j \\ s.t. \|x\|_2 = \sqrt{n} \\ \mathbf{1}^T x = 0\end{aligned}$$

Now we have removed a multiple of $\mathbf{1}\mathbf{1}^T$ [Recall that $\mathcal{A} = A - \frac{p+q}{2}\mathbf{1}\mathbf{1}^T$] so we will drop the constraint $\mathbf{1}^T x = 0$, yielding us

$$\begin{aligned}\max \sum_{i,j} A_{i,j}x_ix_j \\ s.t. \|x\|_2 = \sqrt{n},\end{aligned}$$

the solution is exactly the top eigenvector of A.

Recall that if $\mathcal{A} - \mathbf{E}[\mathcal{A}]$ was a Wigner matrix with independent and identically distributed entries with zero mean and variance σ^2 then its an empirical spectral density would follow the semicircle law and it will essentially be supported in $\left[\frac{-2\sigma}{\sqrt{n}}, \frac{2\sigma}{\sqrt{n}}\right]$

2 Fast Unfolding of Communities in Large Networks

2.1 Notes on Paper

One approach of decomposing a large network of nodes into a smaller set can be done by decomposing networks into sub-units of highly interconnected nodes.

So the problem of community detection requires the partition of a network into communities of densely connected nodes. The community nodes are then only sparsely connected to other communities. The problem then is to find these communities. A few types of algorithms have been proposed, these types include:

1. Devise-types, which detect inter-connected community links and remove them from the network
2. Agglomerative Algorithms merge similar nodes/communities recursively
3. Optimization methods are based on the maximization of an objective function.

A metric for the quality of these partitions is given by the modularity of a partition is a scalar value between -1 and 1 that measure the density of links between members of a community, and the links between communities. In the case of a weighted network:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{i,j} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

where A is the weight matrix, and $A_{i,j}$ represents the weight of the edge between node i and j, $k_i = \sum_j A_{i,j}$ is the sum of the edges attached to the vertex i, c_i is the community to which the vertex i is assigned and $\delta(x, y) = 1 \iff x = y$ and $\delta(x, y) = 0 \iff x \neq y$. Finally $m = \frac{1}{2} \sum_{i,j} A_{i,j}$.

Exact optimization is a algorthimically hard problem, so approximation methods are necessary for large networks. Clauset et. al. has been the fastest proposed. That consists of recursively merging communities that optimize modularity. However, this produces modularity values lower than other methods like simulated annealing.

Proposed here is an algorithm that produces high modularity partitions of large networks in a short time and unfolds a completely hierarchical community structure for the network. For example, identifying communities in 118 million node networks took 152 minutes.

Phase I:

For a weighted network of N nodes.

1. Assign a different community to each node of the network.

   So, in this initial partition there are as many communities as nodes.

2. For each node i, we'll consider the neighbors j of i and we evaluate the gain of modularity that would take place by removing i from its community and by placing in in the neighbors-community j.

3. For each node this is done, and each neighbor of i, j, so that the one chosen is the one that maximizes modularity and is a positive gain (no loss of modularity).

   Where a breaking rule is used in a tie.

   This is done for every node until no further modularity can be gained, and we'll allow for the possibility of applying this to a single node more than once. This first step terminates when a local maximum is attained; no moves can improve the modularity. **The output depends on the order in which the nodes are considered.**

Preliminary tests show that the ordering of nodes doesn't have a significant influence on the modularity, but may effect the computation time (don't know why).

For example taking the natural ordering given by the community structure itself doesn't have a clear improvement.

The efficiency is due to an easy calculation for modularity gain:

$$\Delta Q = \left[\frac{\sum_{in}+2k_{i,in}}{2m} - \left(\frac{\sum_{tot}+k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)\right]$$

- $\sum_{in}$ is the sum of the weights of the links inside C
- $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C
- k_i is the sum of the weights of the links incident to node i
- $k_{i,in}$ is the sum of the weights of the links from i to nodes in C
- m is the sum of the weights of all links in the networks.

'A similar expression is used in order to evaluate the change of modularity when i is removed from the community.' We can calculate the change in modularity by removing i from its community and then by moving it into a neighbouring community.

Phase II:

Build a new network by building a new network whose nodes are the communities from Phase I. The weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding 2 communities. Links between nodes of the same community lead to self-loops for this community in the new network.

Once this is completed, we can reapply Phase I.

One iteration of this is a 'pass'.

We iterated this until there is no changes and a maximum of modularity is attained.

The height of the hierarchy that is constructed depends on the number of passes and is usually small. Shown below.

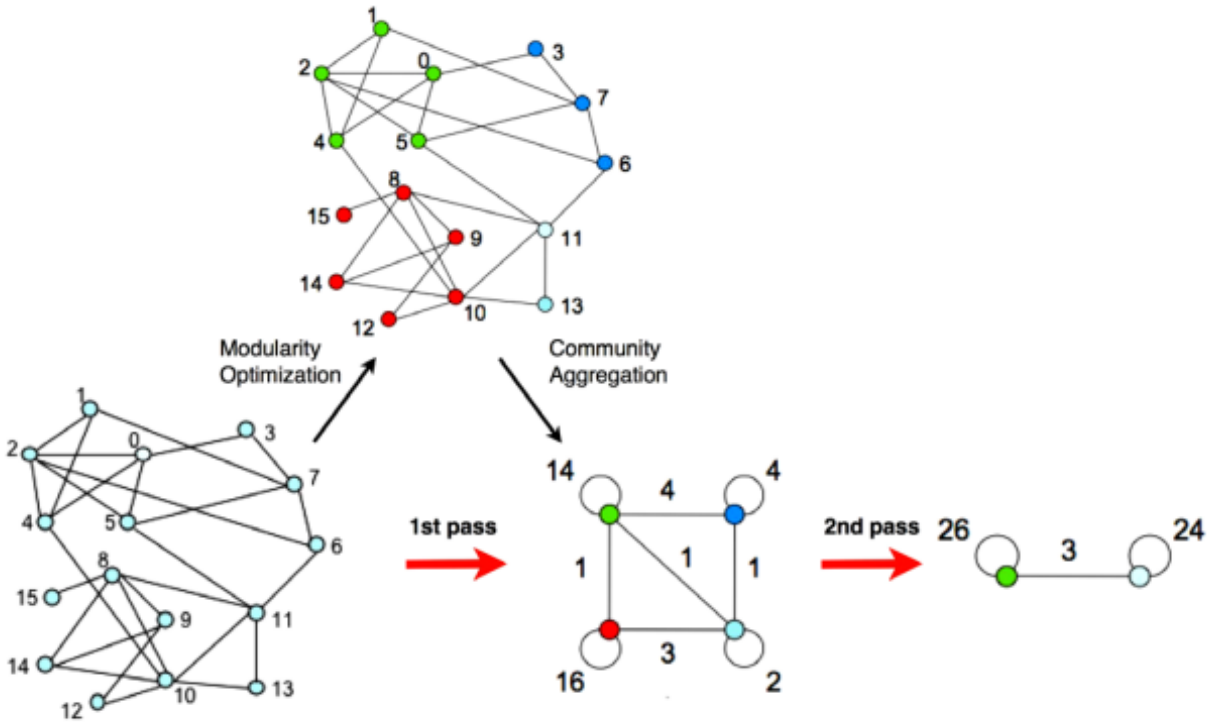

Figure 1. Visualization of the steps of our algorithm. Each pass is made of two phases: one where modularity is optimized by allowing only local changes of communities; one where the communities found are aggregated in order to build a new network of communities. The passes are repeated iteratively until no increase of modularity is possible.

1. In this, we establish a partition by some labelling of the graph.
2. And the second pass, finds the global maximum of modularity where cliques are combined into groups of 2.
3. However, the intermediate steps might hold useful information, and can be saved to allow for a kind of zooming into the network.

This algorithm has advantages, it's intuitive, and easy to implement, and the outcome is 'unsupervised.

Modularity optimization fails to identify communities smaller than a certain scale, thereby inducing a resolution limit on the community detected by a pure modularity optimization approach.

Community in Social and Biological Networks

This is relevant to Phase I, which involves the displacement of single nodes from another community to another. So the likelihood that two communities will merge from this 1-on-1 movement of nodes is very low. But is totally possible in Phase II.

2.2 Summary of Algorithm

Note 1: Summary of Algorithm

Consider a non-directed weighted graph of N-nodes.
Phase I:

1. (Initialization)

   Assign each node its own community (i.e every node i is labeled community i)

2. (Modularity Testing)

   Pick a starting point, consider the effect of removing node i from community i and placing it in one of the neighboring-communities j. Does this increase modularity?

3. (Modularity Optimization) Do this for each neighbor j, and move i to the community j that optimizes the increase to modularity.

4. (Tie Breaking) A tie breaking rule is employed, if a tie occurs

5. (Loop) Start this again at node $i + 1$.

6. (Terminate) Stop this once no modularity can be gained.

Phase II:

1. Create a weighted non-directed graph determined by communities

2. For the inter-connected weights inside the community, sum these to get a self-loop

3. For all the connections to the other communities, sum these, and the total is the weight of the link joining the two communities.

4. Repeat Phase I for this new graph.

Continue this process until no further modularity is gained.
For each graph created (Phase II), save these, as these might be useful intermediate graphs and a full collection of these allows for a zooming-in to a community.

3 Community Structure in Social and Biological Networks (Girvan, Newman 2001)

3.1 Notes on Paper

3.1.1 Introduction

Large Networks, among social situations, usually have a handful of common behaviors/properties:

- Complex networks of nodes, often exhibit a common behavior of finding the average distance between vertices in a network is short (scaled logarithmically)
- Many vertices usually have a low degree with a small number having high degree. The distribution of which is usually a power-law or exponential distribution.
- Clustering or network transitivity, which is the property that two vertices that are neighbors share a common third neighbor. That is, two of your friends are more likely to know each other than complete strangers. This is quantified by:

$$C = 3 \times \frac{\text{Number of Triangles on the Graph}}{\text{Number of Connected Triples of vertices}}.$$

  In a typical network, this is usually in $[0.1, 0.5]$.

We'll be looking at subsets of vertices that are specifically dense in themselves, and have sparse connections outside of these subsets are less dense. Many applications, including identifying friend groups, working groups inside of research communities, and metabolic network connections. We'll be looking at problems where the communities are already known.

3.1.2 Detecting Community Structure

- (Hierarchical Clustering)

  Calculate a weight matrix $W_{i,j}$ for every pair i, j of vertices in the network, that is a measure of how closely connected the vertices are. Then add these edges to the n-vertices of the network, these produce the connections. The large connected components are considered the communities.

  So since these are proper subset we're able to represent these connections with a tree:

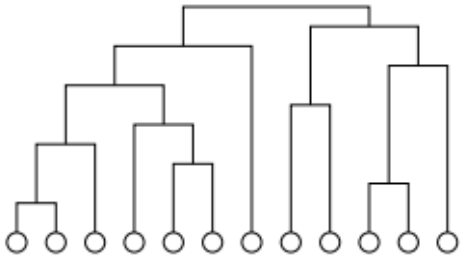

FIG. 2: An example of a small hierarchical clustering tree. The circles at the bottom of the figure represent the vertices in the network and the tree shows the order in which they join together to form communities for a given definition of the weight W_{ij} of connections between vertex pairs.

Slices of this tree at different levels then identifies certain communities, hence the hierarchy. These graphs are called 'dendrograms' in the sociological literature.

The measure of the weights has many ways to calculate these: One is a node-independent path between vertices.

Definition 1: T

at is two paths which connect the same pair of vertices are said to be **node-independent** if they share none of the same vertices other than their final and initial vertices.

Theorem 1: T

e number of node-independent paths between vertices i and j in a graph is equal to the minimum number of vertices that need to be removed from the graph in order to disconnect i and j from one another.

This gives a measure of the robustness of the network to deletion of nodes.

Another method of determining the weight matrix $W_{i,j}$ between the vertices is to count all the total number of paths that run between them (all paths, not just node-independent ones). This number is infinite though, one typically weights the paths of length l by a factor α^l with $\alpha << 1$, so that the weighted count of the number of paths converge. So long paths count for less than short paths.

Let $\mathbf{A}$ be the adjacency matrix of the network such that $A_{i,j} = 1$ if vertices i and j share an

edge and 0 otherwise. Then:

$$\mathbf{W} = \sum_{l=0}^{\infty} (\alpha \mathbf{A})^l = [\mathbf{I} - \alpha \mathbf{A}]^{-1}.$$

So we need $\alpha << \frac{1}{\max_i \lambda}$, where $\mathbf{A}v = \lambda v$.

The problem arises in both of these that peripheral nodes, nodes attached by a single edge to the rest of the network, often remain alone.

We propose a new method that alleviates these problems. We'll focus not on weights but those edges which are least central, the edges most "between" communities. Rather than constructing communities by adding the strongest edges to an initially empty vertex set, we construct them my progressively removing edges from the original graph.

Vertex 'betweenness', has been defined by Freeman, as the centrality of a vertex i as defined as the number of shortest paths between pairs of other vertices which run through i. It measures the influence of that one node to the overall flow of information in the network.

We introduce a similar concept for 'edge betweenness' of an edge as the number of shortest paths between pairs of vertices that run along it. If multiple shortest paths between a pair of vertices, each path is given equal weight such that the total weight of all paths is unity. If an edge joins two communities, then they will have high edge betweenness.

The algorithm we propose for identifying communities is simply stated as follow:

1. Calculate the betweenness for all edges in the network
2. Remove the edge with the highest betweenness
3. Recalculate betweenness for all edges affected by the removal
4. Repeat from Step 2 until no edges remain.

Detailed in "M. E. J. Newman, Scientific collaboration networks: II. Shortest paths, weighted networks, and centrality. Phys. Rev. E 64, 016132 (2001)" which calculates the betweenness for all m edges in a graph of n vertices in $O(mn)$ time. Since this calculation is repeated for each edge, it runs in worst-case time $O(m^2 n)$. With the removal of each edge, we only have to recalculate the betweennesses of those edges that were affected by the removal, which is at most only those in the same component as the removed edge. This is worst when the network is strong community structure.

3.1.3 Tests of the Method

1. Constructing a graph with 128 vertices, each with degree 16. These were divided into 4 communities, with some number z_{in} of each vertex's 16 neighbors made to randomly

choose members of its own community and the remaining $z_{out} = z - z_{in}$ made to random members of other communities.

This worked 100% of the time when $z_{out} \leq 6$.

2. A real life example was applied to Zachary's Karate Club.

   Where 34 members of a karate clue were followed over a 2 year period. The algorithm divides the 34 into 2 groups of people in the 2 karate clubs. We were able to reproduce the breakdown observed in reality by this algorithm.

4 Computing communities in large networks using random walks (Pons, Latapy)

4.1 Notes on Paper

4.1.1 Introduction

The problem of community detection is that of finding locally dense subgraphs of globally sparse graphs. These subgraphs are called **communities**.

Formally the problem is that of considering a partition of

$$P = \{C_1, \ldots, C_k\}$$

of the vertices of a graph $G = (V, E)$, where good community structure means:

$$(\forall i, C_i \subset V)$$

if the portion of internal edges of C_i is high compared to the proportion of edges between the C_i's. Formally this is modularity:

See:

- 'Fast Algorithm for Detecting Community Structure in Networks.'(Newman 2004)
- 'Finding and Evaluating Community Structure in Networks'(Newman, Girvan 2004)

In this paper we'll be letting:

$$|V| = n \qquad |E| = m.$$

And each vertex has a self-loop, and let G be connected, if not we just consider the unconnected components as different graphs.

Computing communities in large networks using random walks (Pons, Latapy)

The idea behind our approach is that random walks tend to get trapped within densely connected parts of a community. This algorithm exceeds in hierarchical clustering algorithms. One obtains a way a hierarchical community structure that may be represented by a *dendrogram*.

We propose an algorithm that computes the community structure in time $\mathcal{O}(mnH)$ where H is the height of the corresponding dendrogram. Worst case that's $\mathcal{O}(mnn)$, most real world networks are sparse with $m = \mathcal{O}(n)$ and balanced graphs will have $H = \mathcal{O}(\log(n))$.

While this is a classical problem of graph partitioning, we need to know the size of the partitions and number of cuts a-priori. So the problem of community detection deals with this. One recent approach is the divisive approach proposed by Girvan and Newman, where the edges with the largest 'betweenness' (# of shortest paths passing through an edge) are removed in order to split the hierarchy of the graph into communities. The algorithm has run time $\mathcal{O}(m^2n)$.

Similar algorithms have been proposed by: "Defining and Identifying Communities in Networks (Radicchi, et.al. 2004)" and "Method to Find Community Structures Based on Information Centrality (Forunata, et.al 2004)" Radicchi uses a local quantity (the number of loops of a given length containing an edge) to choose the edges to remove and runs in $O(m^2)$. Forunata use a more complex notation of information centrality that gives better results but poor run time $0(m^3n)$.

The Agglomerative Algorithm uses hierarchical clustering, where vertices are grouped together to form communities. Newman and Girvan in 'Finding and Evaluating Community Structure in Networks' used this with a Greedy algorithm and poses the problem as an optimization problem over Modularity. This runs in $\mathcal{O}(mn)$ time and in:

- 'Finding Community Structure in Very Large Networks'(Clauset, et.al. 2004)

This was improved to $\mathcal{O}(mH\log(n))$ time.

Donetti and Munoz in:

1. 'Detecting Network Communities: A New Systematic and Efficient Algorithm'(Donetti, Munoz 2004)

Use a hierarchical clustering method they use the eigenvectors of the Laplacian matrix of the graph to measure the similarities between the vertices. The demanding part of this is computing eigen-values and takes $\mathcal{O}(n^3)$ for a sparse matrix.

Some miscellaneous algorithms:

1. 'Finding Communities in Linear Time: A Physics Approach'(Wu, Huberman 2004)
2. 'Hub-Based Community Finding'(Fontoura Costa 2004)

Computing communities in large networks using random walks (Pons, Latapy)

3. 'Detecing Fuzzy Community Structures in Complex Networks with a Pott Model'(Reichardt, Bornholdt 2004)
4. 'A Local Method for Detecting Communities' (Bagrow, Bollt 2005)
5. 'Finding Local Community Structure in Networks'(Clauset 2005)
6. 'Community Detection in Complex Networks Using Extermal Optimization'(Duch, Arenas 2005)

4.1.2 Preliminaries on Random Walks

Definition 2: Adjacency Matrix/Degree

The Adjacency Matrix A of a graph G:

$$A_{i,j} = \begin{cases} 1 & \text{if i and j are connected by an edge} \\ 0 & \text{otherwise} \end{cases}$$

So that the degree of vertex i is:

$$d(i) = \sum_j A_{i,j}$$

In general we can take $A_{i,j} \in \mathbb{R}^+$.

Computing communities in large networks using random walks (Pons, Latapy)

Definition 3

A Discrete Random Walk on a graph G is described in:

- 'Reversible Markov Chains and Random Walks on Graphs'(Aldous, Fill)
- 'Random Walks on Graphs: A Survey'(Lovasz 1996)

Generally this is walk, where at each vertex a random and uniform decision is made among neighbors. The vertices visited by this is a Markov Chain, the probability of transition on a vertex i to vertex j is then:

$$P_{ij} = \frac{A_{i,j}}{d(i)}.$$

This is the transition matrix P of random walk processes.
We also have a transition matrix:

$$P = D^{-1}A$$

where $D \sim Diag\{d(1), \ldots, d(i), \ldots\}$.

So that the probability of going from i to j through a random walk of length t is exactly:

$$(P^t)_{ij}.$$

Theorem 2

When the length t of a random walk starting at vertex i tends towards infinity, the probability of being on a vertex j only depends on the degree of vertex j:

$$\forall i \qquad \lim_{t\in\infty} P_{ij}^t = \frac{d(j)}{\sum_k d(k)}$$

Theorem 3

The probabilites of going from i to j and from j to i through a random walk of a fixed length t have a ration that only dependes on the degrees $d(i)$ and $d(j)$:

$$\forall i \forall j \qquad d(i)P_{ij}^t = d(j)P_{ji}^t.$$

Proof. This property can be written as the matricial equation

$$DP^tD^{-1} = (P^t)^T = P^t(\text{P is symmetric}).$$

By using the fact that $P = D^{-1}A$ and the symmetry of D and A:

$$DP^tD^{-1} = D(D^{-1}A)^tD^{-1} = (AD^{-1})^t = (A^T(D^{-1})^T)^t = ((D^{-1}A)^T)^t = (P^t)^T.$$

□

4.1.3 Comparing Vertices Using Short Random Walks

To first group vertices into communities we need a measure of community structure.

So for a random walk on G with length t, recall the probability of getting from i to j in t-Steps is: P^t_{ij}. We need t to be large enough to discern community structure from its walk, but not long enough that it becomes independent of the degrees of the vertices. We know that $P^t_{i,j}$ and $P^t_{j,i}$ encode the same information by Property 2. That is the community structure near vertex i is determined by the $n-$probabilities of $(P^t_{i,k})_{1\leq k\leq n}$, which is the i^{th} row of P^t: denote P^t_{i*}. To compare a vertex i with j we want:

1. If i and j are in the same communities, the probability $P^t_{i,j}$ will surely be high. But this isn't a sufficient condition for i and j to be in the same community.
2. The probability $P^t_{i,j}$ is influence by the degree $d(j)$ because the walker has a high probability of going to high degree vertices
3. Thus we say that i and j are in the same community if $P^t_{i,k} \simeq P^t_{j,k}$ for all k.

Definition 4

Let i and j be two vertices in the graph and:

$$r_{i,j} = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{ik} - P^t_{jk})^2}{d(k)}} = \|D^{-1/2}P^t_{i*} - D^{-1/2}P^t_{j*}\|$$

where $\|\cdot\|$ is the Euclidean Norm in $\mathbb{R}^n$.
This is just the L^2 distance between the probability distributions:

$$P^t_{i*} \qquad P^t_{j*}.$$

This just dependent on t and so we will denote:

$$r_{i,j} = r_{i,j}(t).$$

Suppose that the random walks starting node is determined randomly and uniformly.

Then

Definition 5

The probaility P^t_{Cj} to go from community C to vertex j in t steps:

$$P^t_{Cj} = \frac{1}{|C|} \sum_{i \in C} P^t_{i,j}.$$

We can use this to define a probability vector $P^t_{C.}$:

Definition 6

Let $C_1, C_2 \subset V$ be two communities. We define the distance $r_{C_1C_2}$ between these two communities as:

$$r_{C_1C_2} = \|D^{-1/2}P^t_{C_1.} - D^{-1/2}P^t_{C_2.}\| = \sqrt{\sum_{k=1}^{n} \frac{(P^t_{C_1k} - P^t_{C_2k})^2}{d(k)}}.$$

This can be used with $C_1 = \{i\}$ and $C_2 = \{j\}$ for the 'distance' between two nodes. And vertex to community:

$$C_1 = C \qquad C_2 = \{i\}.$$

Theorem 4

The distance r is related to the spectral properties of the matrix P by:

$$r^2_{i,j} = \sum_{\alpha=2}^{n} \lambda^{2t}_\alpha (v_\alpha(i) - v_\alpha(j))^2$$

where $(\lambda_\alpha)_{1 \leq \alpha \leq n}$ and $(v_\alpha)_{1 \leq \alpha \leq n}$ are respectively the eigenvalues and right eigenvectors of the matrix P.

Computing communities in large networks using random walks (Pons, Latapy)

Lemma 1

The eigenvalues of P are real and satisfy:

$$1 = \lambda_1 > \lambda_2 \geq \ldots \geq \lambda_n > -1$$

Moreover, there exists an orthonormal family of vectors $(s_\alpha)_{1\leq\alpha\leq n}$ such that each vector $v_\alpha = D^{-1/2}s_\alpha$ and $u_\alpha = D^{1/2}s_\alpha$ are respectively a right and a left eigenvector associated to the eigenvalue λ_α :

$$\forall\alpha, Pv_\alpha = \lambda_\alpha v_\alpha \qquad \text{and} \qquad P^T u_\alpha = \lambda_\alpha u_\alpha$$

$$\forall\alpha, \forall\beta, v_\alpha^T u_\beta = \delta_{\alpha\beta}$$

Proof of Lemma. The matrix P has the same eigenvalues as its similar matrix $S = D^{1/2}PD^{-1/2} = D^{-1/2}AD^{-1/2}$. The matrix S is real and symmetric, so its eigenvalues λ_α are real. P is a stochastic matrix

$$\sum_{j=1}^{n} P_{i,j} = 1$$

so its largest eigenvalue is $\lambda_1 = 1$.

The graph G is connected and primitive (the gcd of the cycle lengths of G is 1, dues to the loops on each vertex), therefore we apply the Perron-Frobenius theorem which implies that P has a unique dominant eigenvalue. Therefore we have:

$$|\lambda_\alpha| < 1 \qquad 2 \leq \alpha \leq n.$$

The symmetry of S implies that there also exists an orthonormal family s_α of eigenvectors of S satisfying $\forall\alpha, \forall\beta, s_\alpha^T s_\beta = \delta_{\alpha\beta}$ (where $\delta_{\alpha\beta} = 1$ if $\alpha = \beta$ and 0 otherwise). We then directly obtain that the vectors $v_\alpha = D^{-1/2}s_\alpha$ and $D^{1/2}s_\alpha$ are respectively a right and a left eigenvector of P satisfying $u_\alpha^T v\beta = \delta_{\alpha\beta}$. □

Proof of Theorem. By lemma 1 it makes it possible to write a spectral decomposition of the matrix P :

$$P = \sum_{\alpha=1}^{n} \lambda_\alpha v_\alpha u_\alpha^T \qquad P^T = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha u_\alpha^T \qquad \Longrightarrow\ P_{ij}^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha(j).$$

When $t \to \infty$, all the terms $\alpha \geq 2$ vanish. It's then easy to show that the first right eigenvector v_1 is constant. By normalizing we have $\forall i,\ v_1(i) = \dfrac{1}{\sqrt{\sum_k d(k)}}$ and $\forall j, u_1(j) =$

$\frac{d(j)}{\sqrt{\sum_k d(k)}}$. We obtain Property 1:

$$\lim_{t\to\infty} P_{ij}^t = \lim_{t\to\infty} \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha(j) = v_1(i)u_1(j) = \frac{d(j)}{\sum_{k=1}^{n}} d(k).$$

Now we obtain the expression of the probability vector P_{i*}^t:

$$P_{i*}^t = \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) u_\alpha = D^{1/2} \sum_{\alpha=1}^{n} \lambda_\alpha^t v_\alpha(i) s_\alpha.$$

We put this formula into the second definition of r_{ij} given in Equation 4.1.3. Then we use the Pythagorean theorem with the orthonomral family of vectors $(s_\alpha)_{1\leq\alpha\leq n}$ and we remember that the vector v_1 is constant to remove the case $\alpha = 1$ in the sum.

Finally we have:

$$r_{ij}^2 = \| \sum_{\alpha=1}^{n} \lambda_\alpha^t (v_\alpha(i) - v_\alpha(j)) s_\alpha \|^2 = \sum_{\alpha=2}^{n} \lambda_\alpha^{2t} (v_\alpha(i) - v_\alpha(j))^2.$$

□

This relates the work of spectral methods being employed to look at community structure in: [41], [40,22] and [10] back to random walks:

41 : "Diffusion on Complex Networks : A way to probe their large-scale topological structures."(Simonsen, et. al., 2004)

40 : "Coarse Grains: The emergence of space and order."(Schulman, Gaveau, 2001)

22 : "Spectral Signatures of Hierarchical Relaxation"(Gaveua, et.al., 1999)

10 : "Detecting Network Communities: A New Systematic and Efficient Algorithm"(Donnetti, Munoz, 2004)

Our method benefits from not having to compute eigenvectors compared to the Spectral methods, which takes $\mathcal{O}(n^3)$ time.

Once the two vectors P_{i*}^t and P_{j*}^t are computed then r_{ij} just takes $\mathcal{O}(n)$ time to calculate.

Theorem 5

Each Probability vector P_{i*}^t can be computed in $\mathcal{O}(tm)$ and space $\mathcal{O}(n)$.

Proof. To compute the vector P_{i*}^t, we multiply t times the vectors P_{i*}^0 $(\forall k, P_{i*}^0(k) = \delta_{ijk})$ by the matrix P. This direct method is advantageous in our case because the matrix P is generally sparse, therefore each product is processed in $\mathcal{O}(m)$. The initialization of P_{i*}^0 is done in $\mathcal{O}(n)$ and thus each of the n vectors P_{i*}^t is computed in time:

$$\mathcal{O}(n + tm) = \mathcal{O}(tm).$$

□

Theorem 6: Approximated Computation

Each probability vector P_{i*}^t can be approximated in time $\mathcal{O}(Kt)$ and space $\mathcal{O}(K)$ with a relative error $\mathcal{O}(\frac{1}{\sqrt{K}})$.

Proof. We compute K random talks of length t starting from vertex i. Then we approximate each probability P_{ik}^t by $\frac{N_{ik}}{N}$ where N_{ik} is the number of walkers that ended on vertex k during the K random walks. The Central Limit Theorem implies that this quantity tends towards P_{ik}^t with speed $\mathcal{O}(\frac{1}{\sqrt{K}})$ when K tends towards infinity. Each random walk computation is done in time $\mathcal{O}(t)$ and constant space hence the overall computation is done in time $\mathcal{O}(Kt)$ and space $\mathcal{O}(K)$. □

5 Data Mining

5.1 Introduction

Definition 7: Clustering Coefficient

The cluster coefficient for a node v_i, which has d_i neighbors, and k_i edges among those neighbors, then the clustering coefficient is:

$$C_i = \begin{cases} \dfrac{k_i}{d_i(d_i-1)/2} & \text{if } d_i > 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In the case of simple graphs $k_i = d_i$.

Networks with community structure tend to have a higher average clustering coefficients then random networks.

Common challenges and advantages that arise with community detection, and that are explored in this book:

- Community detection allows for simplification of networks, and instead of problem solving on a node basis, allows for problem solving on a group basis
- Scalability to large networks (Flake et.al. 2000; Gibson et.al, 2005; Dourisboure et. al, 2007; Andersen and Lang, 2006)
- Heterogeneous interactions, e.g. YouTube comments/likes/dislikes/shares (Zeng et. al, 2002; Java et. al, 2008; Tang et. al. 2008, 2009)
- Temporal evolution of networks (Backstrom et. al., 2006; Palla et. al.,2007; Asur et. al., 2007; Tang et. al., forthcoming)
- Link Prediction e.g. friend suggestions (Liben-Nowell and Kleinber, 2007)
- Collaborative Filtering ((Breese et.al. 1998)
- Network-Based Classification, which is the inferring of behaviors of members of a community based on general community behavior.

Degree centrality is used to measure the importance of one node to the connectedness of a graph.

Definition 8: Degree Centrality

The importance of a node is determined by the number of nodes adjacent to it. The larger the degree of one node, the more important the node is. Node degrees in most social media networks follow a power law distribution, i.e., a very small number of nodes have an extremely large number of connections. Those high-degree nodes are naturally have more impact to reach a larger population than the remaining nodes within the same network. Thus, they are considered more important.
The degree centrality of a node v_i is defined to be:

$$C_D(v_i) = d_i = \sum_j A_{ij}.$$

When one needs to compare two nodes in different networks, a normalized degree centrality should be used:

$$C'_D(v_i) = \frac{d_i}{n-1}.$$

Where n is the number of nodes in a network. This is the proportion of nodes in the network adjacent to v_i.

Definition 9: Closeness Centrality

A central node should reach the remaining nodes more quickly than a non-central node. Closeness centrality measures how close a node is to all the other nodes. The measure involves the computation of the average distance of one node to all the other nodes:

$$D_{avg}(v_i) = \frac{1}{n-1} \sum_{j \neq i}^{n} g(v_i, v_j),$$

where n is the number of nodes and $g(v_i, v_j)$ is the geodesic distance between nodes v_i and v_j.

Generally, this is a kind of measure of information dissemination, how long does it take information starting from v_i to reach the whole network.

So that a lower $D_{avg}(v_i)$ means that node is more central to a network. So we define closeness centrality as:

$$C_C(v_i) = D_{avg}^{-1}(v_i) = \frac{n-1}{\sum_{j \neq i}^{n} g(v_i, v_j)}$$

Definition 10: Betweenness Centrality

Node betweenness counts the number of shortest paths in a network that will pass a node. Those nodes with high betweenness play a key role in the communication within a network. The betweenness centrality of a node is defined as:

$$C_B(v_i) = \sum_{v_s \neq v_i \neq v_t \in V, s<t} \frac{\sigma_{st}(v_i)}{\sigma_{st}},$$

where σ_{st} is the total number of shortest paths between nodes v_s and v_t, $\sigma_{st}(v_i)$ is the number of shortest paths between nodes v_s and v_t that pass along node v_i.
This is essentially, the probability that a shortest path between v_s and v_t will pass through v_i. In an undirected network the largest this can be is $C_B(v_i) = \binom{n-1}{2} = (n-1)(n-2)/2$.
So the normalized betweenness centrality is:

$$C'_B(i) = \frac{2C_B(i)}{(n-1)(n-2)}.$$

As you can imagine this takes quite a bit of time to compute, it takes at least $O(mn)$ time to compute, where n is the number of nodes and m is the number of edges. Appendix B has a detailed algorithm on how to implement this.

Definition 11: Eigenvector Centrality

The idea behind this is that one who has many important friends must also be important. In particular:

$$C_E(v_i) \propto \sum_{v_j \in N_i} A_{ij} C_E(v_j)$$

Let $\mathbf{x}$ denote the eigenvector centrality of node from v_1 to v_n. Then this is equivalent to:

$$\mathbf{x} \propto A\mathbf{x}$$

That is:

$$\mathbf{x} = \frac{1}{\lambda} A\mathbf{x} \iff A\mathbf{x} = \lambda\mathbf{x}.$$

Thus $\mathbf{x}$ is an eigenvector of the adjacency matrix, where $\mathbf{x} = \{C_E(v_1), \ldots, C_E(v_n)\}$

Google's Pagerank (Page et.al., 1999) used a variant of this eigenvector centrality. They first

normalized so that each column is normalized to sume to 1:

$$\tilde{A_{ij}} = \frac{A_{ij}}{\sum_i A_{ij}}.$$

So that the entry $\tilde{A_{ij}}$ is the probability of transition from node v_j to node v_i, Google used this as the probability of one user clicking on a web page node v_i after browsing current web page v_j by following the line $e(v_j, v_i)$. In this example the eigenvector was computed by the power method, by repeatedly left-multiplying a non-negative vector $\mathbf{x}$ with $\tilde{A}$ till they reach convergence:

$$A = \begin{bmatrix} 0 & 0.5 & 0.\overline{3} & 0.25 & 0 & 0 & 0 & 0 & 0 \\ 0.\overline{3} & 0 & 0.\overline{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.\overline{3} & 0.2 & 0 & 0.25 & 0 & 0 & 0 & 0 & 0 \\ 0.\overline{3} & 0 & 0.\overline{3} & 0 & 0.25 & 0.25 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0 & 0.25 & 0.25 & 0.\overline{3} & 0 \\ 0 & 0 & 0 & 0.25 & 0.25 & 0 & 0.25 & 0.\overline{3} & 1 \\ 0 & 0 & 0 & 0 & 0.25 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.25 & 0 & 0 \end{bmatrix}.$$

Starting off with $\mathbf{x}^0 = \mathbf{1}$ we get $\mathbf{x} \propto \tilde{A}\mathbf{x}^0, \mathbf{x}^2 \propto \tilde{A}\mathbf{x}$, etc. Typically $\mathbf{x}$ is normalized to unit length. Detailed in (Golub and Van Loan, 1996).

Believe it or not this is actually the least time intensive centrality to measure usually with time complexity of $O(ml)$ where l is the number of iterations and m is the number of edges.

As in friend networks every tie is not the same importance, that is, we there are different levels of ties. There are three main approaches on how to measure these:

1. Analyzing Network Topology
2. Learning From User Attributes
3. Learning From User Activities

1. In the approach of analyzing network topology, we look for 'bridges' between two communities. These are nodes whose removal means a high level of disconnection. The larger the geodesic distance between terminal nodes after an edges removal, the weaker that tie is. That is, an edge's removal resulting a high increase in geodesic distance, then that is a weak tie compared to an edge where the geodesic distance is minimally impacted by it's removal.

   Based on (Onnela et. al. 2007 Easley and Kleinberg 2010) another way to measure this is looking at the neighborhood overlap of terminal nodes. Let N_i denote the friends

of v_i (just the neighborhood of v_i). Given link $e(v_i, v_j)$, the neighborhood overlap is defined as:

$$Overlap(v_i, v_j) = \frac{\text{Number of Shared Friends of Both } v_i \text{ and } v_j}{\text{Number of friends who are adjacent to at least } v_i \text{ or } v_j} = \frac{|N_i \cap N_j|}{|N_i \cup N_j| - 2},$$

-2 omits v_i and v_j from the set $N_i \cup N_j$. In (Onnela et. al, 2007) this correlated well with the total number of minutes spent by two people in a telecommunication network.

2. In Gilbert and Karahalios (2009) an attempt was made to use many different variables such as posts, days since last communication, positive/negative emotion words, age difference, education difference, etc. to determine strength of connections amongst Facebook friends. This linear model achieved a 85 % success rate.

   Xiang et. al. (2010) proposed a weighted graph model, where the score was based on same school attended, work at the same company, live in the same area, etc. As well as interaction feeding into this score. By optimizing the joint probability given user profiles and interaction information this showed a strength over other models.

3. Finally, we get to the idea of learning from sequences of user activities. This relies almost entirely on a static network snapshot. Kossinets et. al. 2008 studied how information disseminates in communication networks, and found that often it did not follow a shortest path between two points, instead followed 'Network Backbones' that are defined to be those ties that are likely to bear the task of propagating the timely information.

5.2 Influence Modeling

One fundamental question in network analysis is that of influence modeling; that is how do we model the spread of new ideas, viral marketing, and overall diffusion. Two basic approaches are the most common **linear threshold modeling** and the **independent cascade model**. Both assume:

- A social network is a *directed graph*, with each actor being a single node;
- Each node starts as active or inactive;
- A node, once activated, will activate his neighboring node;
- Once a node is activated, the node cannot be deactivated.

5.2.1 Linear Threshold Model

This is a very old model, dating back to Granovetter, 1978. It says that an actor will take an action if the number of his friends who have taken the action exceeds a certain threshold. A variant of this model was used by Schelling in 1971 to show tha small preference for one's neighbors to be the same race would lead to racial segregation.

In a linear thresehold model, each node v_i has a thresehold θ_v that is chosen from a uniform distribution $[0, 1]$. The thresehold θ_v is the fraction of friends v_i need to be active in order to activate v_i. Suppose that a neighbor w can influence v_i with strength $b_{v,w}$. W.L.O.G we assume that :

$$\sum_{w \in N_v} b_{w,v} \leq 1.$$

Starting with an initial active set $A_0 \subset V$, the diffusion process will behave deterministically. In each step, the nodes where:

$$\sum_{w \in N_v, w \text{is active}} b_{w,v} \geq \theta_v$$

will now become active.

In a directed graph, it's usually the case that $b_{v,w} \neq b_{w,v}$.

For simplicity, we can start with