Some Community Detection Algorithms

Social Network Analysis

Lê Hồng Phương

<phuonglh@hus.edu.vn>
Data Science Laboratory
Vietnam Institute for Advanced Study in Mathematics

June 12, 2018

Content

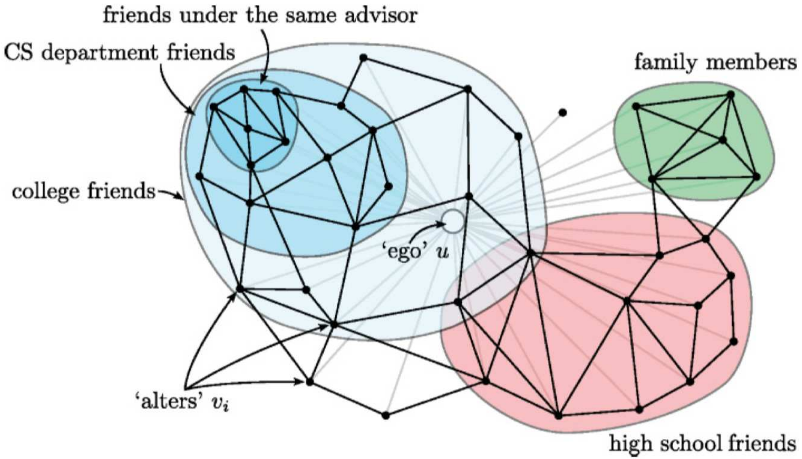
Content

Content

Content

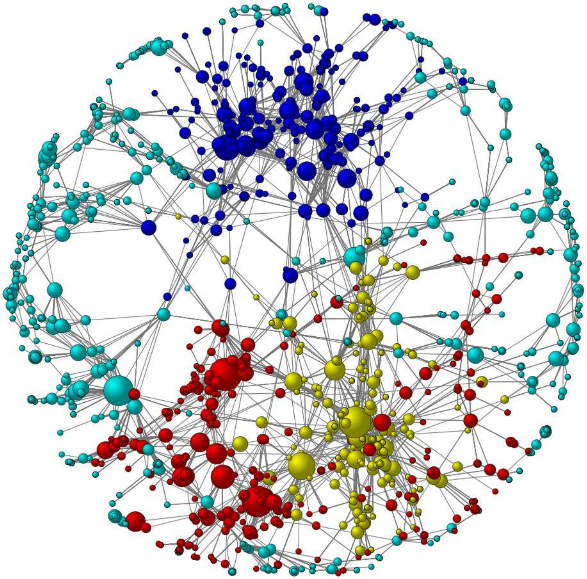
Content

An Ego Network



Julian McAuley and Jure Leskovec, *Discovering Social Circles in Ego Networks*, ACM Transactions on Knowledge Discovery from Data, 2014.

A Social Network



Content

Homophily

- Nodes are connected to one another are more likely to have similar properties.
- Individuals who are linked often share common beliefs, backgrounds, education, hobbies or interests.

Triadic Closure

- The structural version of homophily.
- The inherent tendency of real-world networks to cluster.
- If two individuals in a social network have a friend in common, then it is more likely that they are either connected or will eventually become connected in the future.
- Related to the **clustering coefficient** of the network.

Clustering Coefficient

- Let $G = (N, A)$ be a undirected graph. S_i is the set of nodes connected to node i and $n_i = |S_i|$.
- There are $\binom{n_i}{2}$ possible edges between nodes in S_i.
- The clustering coefficient of node i is the fraction of these pairs that have an edge between them.

$$\eta(i) = \frac{\#(j,k) \in A : j,k \in S_i}{\binom{n_i}{2}}$$

- The **network average clustering coefficient** is the average value of $\eta(i)$ over all nodes in the network.

Power-law Degree Distributions

- A small minority of high-degree nodes continue to attract most of newly added nodes.
- The number of nodes $P(k)$ with degree k is regulated by the power-law degree distribution

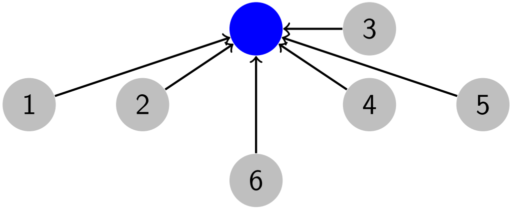
$$P(k) \propto k^{-\gamma}, \quad 2 \leq \gamma \leq 3.$$

Measures of Centrality

- Measures of centrality are defined for *undirected networks*.
- The degree centrality $C_D(i)$ of a node i is $degree(i)/(n-1)$.
- Nodes with higher degree are often hub nodes. They tend to be more central to the network and bring distant parts of the network closer together.
- Major problem: $C_D(i)$ only looks at each local nodes. The overall architecture of the network is ignored at some extent.

Measures of Prestige

- Measures of prestige are defined for directed networks.
- The degree prestige $P_D(i)$ of a node i is $indegree(i)/(n-1)$.
- The number of followers determines the degree prestige of a node.



Closeness Centrality

- Defined for undirected and connected network.
- The degree centrality measure does not consider indirect relationships to other nodes. The **closeness centrality** is more effective in capturing the structure of a network.
- Let $Dist(i, j)$ the shortest path distance between two nodes i and j. The average shortest path distance starting from node i is defined as

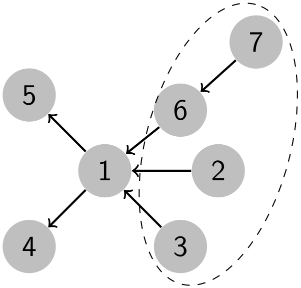
$$AvDist(i) = \sum_j Dist(i, j)/(n-1).$$

- The closeness centrality is defined as $C_C(i) = 1/AvDist(i)$.
- Note that $0 \leq C_C(i) \leq 1$

Proximity Prestige

- Defined for directed networks. Let $Influence(i)$ be the set of nodes that can reach node i with a directed path.
- The value of $AvDist(i)$ is computed with respect to the influence set of i:

$$AvDist(i) = \sum_{j \in Influence(i)} \frac{Dist(j,i)}{|Influence(i)|}.$$



Proximity Prestige

- The influence fraction of node i is defined as

$$InfluenceFraction(i) = \frac{|Influence(i)|}{(n-1)}$$

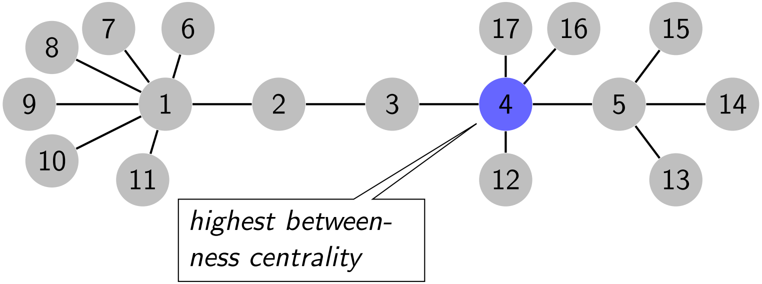
- The proximity prestige $P_P(i)$ is defined as

$$P_P(i) = InfluenceFraction(i)/AvDist(i),$$

- Note that $0 \leq P_P(i) \leq 1$. Higher values indicate higher prestige.

Betweenness Centrality

- Betweenness centrality measures the criticality of a node in terms of the number of shortest paths that pass through it.
- This measures helps determine nodes that have greatest control of the flow of information between other nodes in a social network.
- Let q_{jk} denote the number of shortest paths between nodes j and k.
- Let $q_{jk}(i)$ be the number of these pairs that pass through node i.



Betweenness Centrality

- The fraction $f_{jk}(i) = q_{jk}(i)/q_{jk}$ indicates the level of control that node i has over j and k in terms of regulating the flow of information between them.
- The **betweenness centrality** $C_B(i)$ is the average value of this fraction over all pairs of nodes:

$$C_B(i) = \frac{\sum_{j<k} f_{jk}(i)}{\binom{n}{2}}.$$

- Note that $0 \leq C_B(i) \leq 1$. Higher values indicate better betweenness.

Betweenness Centrality for Edges

- The notion of betweennes centrality for nodes can be generalized to edges by computing the number of shortest paths passing through an edge rather than a node.
- Normally, edges connecting hub nodes have high betweenness.
- Edges that have high betweenness tend to connect nodes from different clusters in the graph.
- These betweenness concepts are used in many community detection algorithms, such as the Girvan-Newman algorithm.

Content

Challenges

- Multi-dimensional clustering methods such as the distance-based k-means algorithm cannot be easily generalized to networks:
  - The distance between pairs of nodes may not provide a sufficiently gine-grained indicator of similarity.
  - It is more important to use the structural properties of real networks (triadic closure properties) in the clustering process.
- In real social networks, the structure is very complicated (overlapping, subsets, etc).

Challenges

- Different parts of the social network have different edge densities.
- The local clustering coefficient in distinct parts of the social network are typically quite different.
- Therefore, a single global parameter choice is not relevant in many network localities, which will lead to unbalanced clusters.

Content

Spectral Clustering

- Use a graph embedding approach: *embed the nodes into a multidimensional space* $\mathbb{R}^d$*.*
- The local clustering structure of the graph is preserved.
- Use a standard k-means clustering algorithm on the distributed representation.

Spectral Clustering

- First, consider the simpler mapping problem with $d = 1$.
- We want to map n nodes of the graph into a vector

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^n,$$

where node i is mapped to a real value $y_i \in \mathbb{R}^1$.

Spectral Clustering

- Suppose that $W = (w_{ij})_{n\times n}$ is the matrix of edge weights, w_{ij} is the weight on the edge (i,j).
- We want that nodes that are connected with high-weight edges to be mapped onto close points on the real line $\mathbb{R}^1$.
- The values of y_i can be determined by minimizing the following objective function:

$$\boxed{J(\mathbf{y}) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(y_i - y_j)^2}$$

- When w_{ij} is large, the points y_i and y_j will be more likely to be closer to one another in the embedded space.

Spectral Clustering

- Denote $\lambda_{ii} \equiv \sum_{j=1}^{n} w_{ij}$ the sum of the weights of the edges incident on node i.
- Denote Λ the diagonal matrix:

$$\Lambda \equiv \begin{pmatrix} \lambda_{11} & & & 0 \\ & \lambda_{22} & & \\ & & \ddots & \\ 0 & & & \lambda_{nn} \end{pmatrix}$$

- Denote $L \equiv \Lambda - W$. The objective function $J(\mathbf{y})$ can be rewritten as

$$\boxed{J(\mathbf{y}) = 2\mathbf{y}^\top L\,\mathbf{y}}.$$

Spectral Clustering

- Since $J(\mathbf{y}) \geq 0, \forall\, \mathbf{y}$, the matrix L is positive semi-definite.
- We want a non-trivial solution where $\mathbf{y} \equiv \mathbf{0}$, therefore we need to impose a scaling constraint, such as:

$$\mathbf{y}^\top \Lambda\, \mathbf{y} = 1.$$

- This is called *normalized spectral clustering*.
- Effect of normalization:
    - *Low-degree nodes tend to clearly pick sides with either large positive or large negative values of* y_i.
    - *High-degree nodes would be embedded closer to central regions near the origin.*

Spectral Clustering

- To solve this constraint optimization problem, we set the gradient of its Lagrangian relaxation to zero:

$$\frac{\partial}{\partial \mathbf{y}} \left[\mathbf{y}^\top L \, \mathbf{y} - \lambda(\mathbf{y}^\top \Lambda \, \mathbf{y} - 1) \right] = \mathbf{0}.$$

- It can be shown that the optimization condition is

$$\Lambda^{-1} L \, \mathbf{y} = \lambda \, \mathbf{y},$$

  that is $\mathbf{y}$ is **an eigenvector** of $\Lambda^{-1}L$, and the Lagrangian parameter λ is an eigenvalue.
- There is a trivial solution: $\lambda = 0$ and $\mathbf{y} \propto (1, 1, \ldots, 1)^\top$ – Every node is mapped to the same point. The second smallest eigenvalue is informative and provides the optimal solution.

Spectral Clustering – Generalization

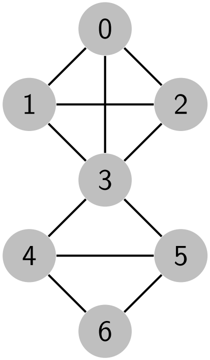
- In the general case, we map each node to a k-dimensional embedding.
- The embedding matrix is $Y_{n\times k} = (\mathbf{y}_1{}^\top, \mathbf{y}_2{}^\top, \dots, \mathbf{y}_n{}^\top)$, where $\mathbf{y}_i \in \mathbb{R}^k$.
- The problem is then to minimize the trace of the $k \times k$ matrix $Y^\top L Y$ subject to the normalization constraints $Y^\top \Lambda Y = I$.
- The optimal solutions for vectors $\mathbf{y}_i$ can be shown to be proportional to the successive directions corresponding to the right eigenvectors of the assymmetric $\Lambda^{-1}L$ **with increasing eigenvalues.**

Spectral Clustering – Generalization

The Spectral Clustering Algorithm

```
Data: $\mathcal{W} = (w_{ij})_{n\times n}$ edge weight matrix of a graph $G$
Result: Spectral representations of nodes
1 compute diagonal matrix $\Lambda$ : $\Lambda_{ii} \leftarrow \sum_{j=1}^{n} w_{ij}$;
2 compute Laplacian matrix $L \leftarrow \Lambda - W$;
3 $A \leftarrow D^{-1} \times L$ ;
4 compute corresponding eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$ and their
    corresponding eigenvectors;
5 sort the eigenvalues in acending order;
6 take top $k$ eigenvectors $[\vec{v}_1, \vec{v}_2, \dots, \vec{v}_k]$ corresponding to top $k$ smallest
    eigenvalues;
7 return $[\vec{v}_1, \vec{v}_2, \dots, \vec{v}_k]$;
```

Spectral Clustering – Example

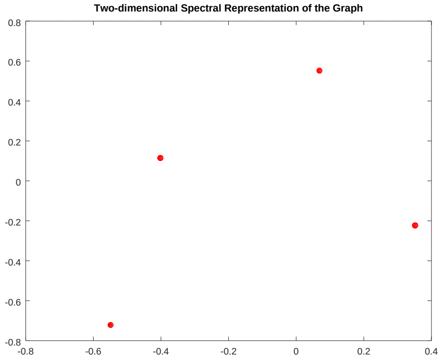


$$W = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Spectral Clustering – Example

$$\vec{\lambda} = \begin{pmatrix} \cancel{0.00000} \\ \mathbf{0.26819} \\ \mathbf{1.15905} \\ 1.33333 \\ 1.33333 \\ 1.33333 \\ 1.57275 \end{pmatrix}$$

$$Z = \begin{pmatrix} 0.351671 & -0.222769 \\ 0.351671 & -0.222769 \\ 0.351671 & -0.222769 \\ 0.068725 & 0.551834 \\ -0.401773 & 0.114726 \\ -0.401773 & 0.114726 \\ -0.549014 & -0.721302 \end{pmatrix}$$



Spectral Clustering

- An equivalent way of setting up the spectral clustering model is to use the related vector $=\Lambda^{1/2}\,\mathbf{y}$.
- This is referred to as the **symmetric version** of the spectral clustering model.
- The equivalent formulation is to optimize

$$J(\mathbf{z}) = \mathbf{z}^\top \Lambda^{-1/2} L \Lambda^{-1/2} \mathbf{z} \text{ subject to } \mathbf{z}^\top \mathbf{z} = 1.$$

Content

Kernighan–Lin Algorithm

- Start with an initial partitioning of the graph into two equal subsets of nodes.
- Iteratively improve this partitioning, until converges to an optimal solution.
- This solution is not guaranteed to be the global optimum.
- How to improve iteratively?

Kernighan–Lin Algorithm

- Determine sequences of exchanges of nodes between partitions that improve the clustering objective function as much as possible.
- The *internal cost* I_i of node i is the sum of weights of edges incident on i, whose other end is present in the same partition as node i.
- The *external cost* E_i of node i is the sum of weights of edges incident on i, whose other end is in a different partition than node i.

Kernighan–Lin Algorithm

- The gain D_i by moving a node i from one partition to the other is the difference
$$D_i = E_i - I_i$$
- The gain J_{ij} of exchanging nodes i and j between two partitions is given by
$$J_{ij} = D_i + D_j - 2w_{ij}$$
- If $J_{ij} > 0$ then there is an improvement of the objective function.

Kernighan–Lin Algorithm

- Perform many sequences of node exchanges, called *epochs*.
- Each epoch has k exchanges ($k \leq n/2$) which is designed to optimize the total gain from the exchanges:
    1. Find the best pair of nodes to exchange with the best gain g_1, mark them;
    2. Recompute values D_j for each node j under the assumption that they will be exchanged eventually;
    3. Repeat: find the next best pair of unmarked nodes to exchange with the best gain g_2, mark them;
    4. Determine k that maximize $G_k = \sum_{t=1}^{k} g_t$;
    5. If $G_k > 0$ then perform the exchange sequences;
- If no epoch with positive gain can be found then the algorithm terminates.

Kernighan–Lin Algorithm

```
   Data: $\mathcal{W} = (w_{ij})_{n\times n}$ edge weight matrix of a graph $G = (N, A)$
1  create random initial partition of $N$ into $N_1$ and $N_2$;
2  repeat
3      recompute $D_i$ for each node $i \in N$;
4      unmark all nodes in $N$;
5      for $t = 1$ to $n/2$ do
6          find unmarked nodes $u_t \in N_1$ and $v_t \in N_2$ with the highest
            exchange gain $g_t = J_{u_t v_t}$;
7          mark $u_t$ and $v_t$;
8          recompute $D_i$ for each node $i$ (as if they are exchanged);
9      end
10     determine $k$ that maximize $G_k = \sum_{t=1}^{k} g_t$;
11     if $G_k > 0$ then
12         exchanges $(u_t, v_t)$ pairs between $N_1$ and $N_2$, for all
            $t = 1, \ldots, k$;
13     end
14 until $G_k \leq 0$;
```

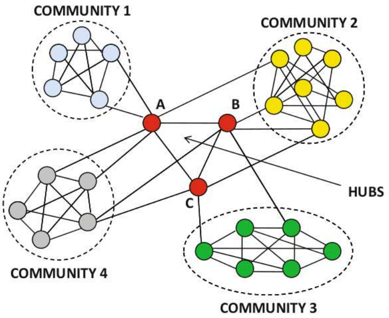
Kernighan–Lin Algorithm

- The Kernighan–Lin algorithm converges rapidly to a local optimum.
- It is usually required fewer than 5 epochs for the algorithm to terminate.
- Since the problem is NP-hard, there is no guarantee on the required number of epochs.
- Variants of the algorithm have been proposed to speed up the the algorithm.

Content

Girvan–Newman Algorithm

The Girvan–Newman algorithm is based on the instuition that edges with high betweenness have a tendency to connect different clusters.



There are a large number of pairwise shortest paths between nodes of different communities passing through these edges.

Girvan–Newman Algorithm

- Top-down hierachical clustering algorithm: creates clusters by successively removing edges with the highest betweenness until the graph is disconnected into the required number of connected components.
- Because each edge removal impacts the betweenness values of some of the other edges, the betweennes values of these edges need to be recomputed after each removal.

Girvan–Newman Algorithm

Girvan–Newman Algorithm

```
  Data: $G = (N, A)$, number of clusters $k$
1 compute the betweenness values of all edges in $A$;
2 repeat
3     remove edge $(i, j)$ from $G$ with highest betweenness;
4     recompute betweenness of edges affected by removal of $(i, j)$;
5 until ($G$ has $k$ components remaining);
6 return $k$ connected components of $G$
```

Girvan–Newman Algorithm

- The main challenge in the Girvan–Newman algorithm is the computation of the **edge betweenness values**.
- The computation of node betweenness values is an intermediary step in the edge betweenness computation.
- Suppose that s is a source node and we consider all shortest paths originating from s.
- Let
  - $B_s(i)$ be the node betweenness centrality of node i, and
  - $b_s(i, j)$ be the edge betweenness centrality of edge (i, j)

  that corresponds to the set of all shortest paths starting from the source node s.
- These two components can then be added over all possible source nodes to compute the overall betweenness centrality values.

Girvan–Newman Algorithm

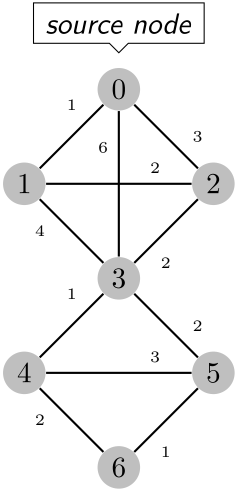
- The first step is to create the *tight* graph which contains *tight edges*.
- An edge is called tight edge if it lies on *at least* one shortest path from node s to some other node.
- The value $b_s(i, j)$ of an edge (i, j) for a particular source node s can be nonzero only if that edge is tight for the source node.
- Let $SP(j)$ is the shortest distance from the source node s to node j.

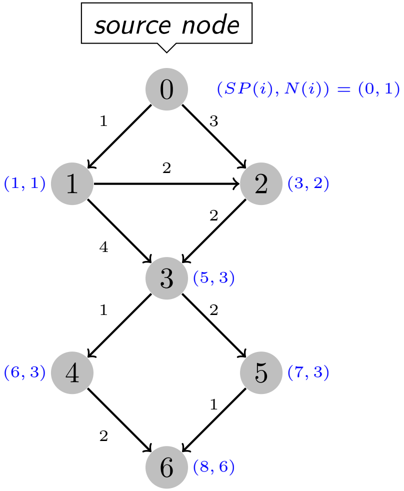
Girvan–Newman Algorithm

- In order for an edge (i, j) to be tight, the following condition has to hold:
  $$SP(j) = SP(i) + c_{ij},$$
  where c_{ij} is the length of edge (i, j).[1]
- The directed subgraph $G^s = (N, A^s)$ of tight edges is constructed, where the direction of the edge (i, j) is such that $SP(j) > SP(i)$.
- Let $N_s(j)$ be the number of shortest paths from the source node s to a given node j. On the tight graph, we have
  $$N_s(j) = \sum_{i:(i,j)\in A^s} N_s(i),$$
  where at the source node $N_s(s) = 1$.

[1] Normally, $c_{ij} = 1/w_{ij}$.

Girvan–Newman Algorithm





Girvan–Newman Algorithm

- The algorithm performs a breadth first search of G^s, starting with the source node s.
- The number of paths to each node is computed as the sum of the paths to its ancestors.
- The next step is to compute the betweenness centrality for both nodes and edges starting at the source node s.

Girvan–Newman Algorithm

- Let $F_{sk}(i)$ be the fraction of shortest paths between nodes s and k that pass through node i.
- Let $f_{sk}(i,j)$ be the fraction of shortest paths between nodes s and k that pass through edge (i,j).
- We have

$$B_s(i) = \sum_{k \neq s} F_{sk}(i)$$

$$b_s(i) = \sum_{k \neq s} f_{sk}(i,j)$$

Girvan–Newman Algorithm

- G^s is used to compute betweenness centrality values by using recursive relationships between $B_s(i)$ and $b_s(i,j)$:

$$B_s(j) = \sum_{i:(i,j)\in A^s} b_s(i,j)$$

$$B_s(i) = 1 + \sum_{j:(i,j)\in A^s} b_s(i,j)$$

- With the source node s: $B_s(s) = 0$.
- Note that $F_{si}(i) = 1$.

Girvan–Newman Algorithm

- The nodes and edges of G^s are processed “bottom up”, starting at the nodes without any outgoing edges; $B_s(j) = F_{sj}(j) = 1$.
- The score $B_s(i)$ of a node i is finalized only after the scores on all its outgoing edges have been finalized.
- The score $b_s(i, j)$ of an edge (i, j) is finalized only after the score $B_s(j)$ of node j has been finalized.

Girvan–Newman Algorithm

The algorithm iteratively updates scores of nodes and edges in the bottom-up traversal as follows:

- **Edge Betweenness Update**:

$$b_s(i,j) = \frac{N_s(i)B_s(j)}{\sum_{k:(k,j)\in A^s} N_s(k)}$$

- **Node Betweenness Update**:

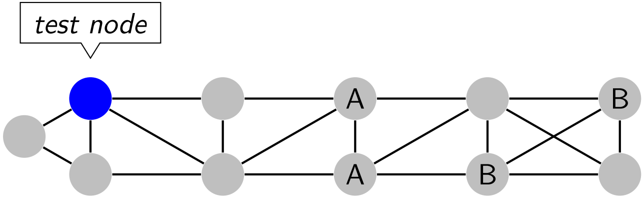
$$B_s(i) = 1 + \sum_{j:(i,j)\in A^s} b_s(i,j)$$

The entire procedure is repeated over all source nodes and the values are added up. The unnormalized values of the node and edge betweenness range from 0 to $n \times (n-1)$.

Content

Collective Classification

- Many nodes in the graph have associated contents (or labels, or properties).
- **Collective classification** algorithms use both node content and graph structure to to classify nodes.
- Note that node labels are sparse: many nodes are unlabeled.



- The test node is generally closer to instances of A rather than B, but there is no labeled node *directly* connected to the test node.

Iterative Classification Algorithm

- The **Iterative Classification Algorithm** (ICA) has the ability to use content associated with the nodes for classification.
- Let $\vec{X}_i$ is the content available at the node i in the form of a multi-dimensional feature vector.
- n is the total number of nodes, n_t is the total number of test nodes.
- The first important step of the ICA is to derive a set of **link features** in addition to the content features in $\vec{X}_i$.

Collective Classification

- For each node, we compute the distribution of the classes in the immediate neighborhood of the node.
- Each class will generate a link feature, which is the fraction of incident nodes belong to that class.
- We can also derive other link features based on structural properties of the graph such as *the degree of the node*, *PageRank values*, *connectivity features*, etc.
- Once we have the link and content features, we can use a base classifier $\mathcal{A}$, such as Naive Bayes classifier to compute the likelihood that it belongs to a particular class:

$$P(y_i | \underbrace{\langle \text{content features}, \text{link features} \rangle}_{\mathbf{x}_i})$$

Collective Classification

- The ICA uses an iterative approach for augmenting the training data set.
- In each iteration, n_t/T test node labels are made “certain” by the approach. The test nodes for which the classifier exhibits the highest class membership probabilities are selected to be made final.
- The labeled test nodes can then be added to the training data. The classifier is retrained by extracting the link features again.
- The approach is repeated until the labels of all nodes have been made final.

Collective Classification

Iterative Classification Algorithm

```
  Data: $G = (N, A)$, number of iterations $T$, base classifier $\mathcal{A}$
1 for $t \leftarrow 1$ to $T$ do
2     extract link features at each node with current training data;
3     train classifier $\mathcal{A}$;
4     predict labels of test nodes;
5     make labels of most “certain” $n_t/T$ test nodes final;
6     add these nodes to training data, remove them from test data;
7 end
```

Content

Summary

- Some basic concepts in social network analysis
- Challenges in community detection
- Some typical algorithms for community detection:
    1. Spectral clustering
    2. Kernighan–Lin algorithm
    3. Girvan-Newman algorithm
    4. Iterative classification algorithm