Introduction to Social Network Analysis

Lê Hồng Phương
Data Science Laboratory
<phuonglh@hus.edu.vn>

December 5, 2018

Content

1 Introduction

2 Basic Concepts

3 Community Detection
   - Spectral Clustering
   - Kernighan–Lin Algorithm
   - Girvan–Newman Algorithm
   - Collective Classification

4 Influence Models
   - Population-based Model

5 Summary

About Speaker

- Dr. Lê Hồng Phương, Director of Data Science Laboratory, Hanoi University of Science, Vietnam National University
- Associate member of DS Labs, Vietnam Institute for Advanced Study in Mathematics (VIASM)
- PhD in Computer Science, Université de Lorraine, France, 2010
- Senior research scientists at FPT Corp and Al+ (Japan)
- http://mim.hus.vnu.edu.vn/phuonglh/
Networks and Communities

We often think of networks being organized into **modules**, **clusters**, **communities**. The goal is to find densely linked clusters.

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

A Social Network

- **Facebook**, **Twitter**, **Google+** (profiles, friendships, etc)
- **Telephone networks**:
  - nodes = phone numbers (individuals)
  - edge = a call between two phones in a fixed period of time
  - weight = number of calls made
- **Email networks**:
  - nodes = individuals
  - edge = there were emails in both direction (to avoid spammers)
Varieties of Social Networks

- **Collaboration networks:**
  - nodes = individuals who have published research papers
  - edge = two co-authors of one or more papers
  - community = authors working on a particular topic

- **Wikipedia editor networks:**
  - people who edit Wikipedia articles and the articles they edit
  - two editors are connected if they have edited an article in common
  - communities are group of editors that are interested in the same subject.

- **Wikipedia article networks:**
  - connect articles if they have been edited by the same person;
  - communities of articles on similar or related subjects.

Varieties of Social Networks

- **Information networks (documents, web graphs, patents)**
- **Infrastructure networks (roads, planes, water pipes, powergrids)**
- **Biological networks (genes, proteins, food-webs of animals eating each other)**
- **Product co-purchasing networks (recommender systems)**

Graphs

**Definition**

A graph is an ordered pair \( G = (V, E) \) comprising a set \( V \) of nodes and a set \( E \) of edges.

\( V \) and \( E \) are usually taken to be finite.

- the **order** of a graph is \(|V|\), the number of vertices.
- the **size** of a graph is \(|E|\), the number of edges.
- the **degree** of a vertex is the number of edges that connect to it.

Weighted Graphs

- A graph is a **weighted** graph if a number (weight) is assigned to each edge.
- Such weight might represent costs, lengths or capacities... depending on the problem at hand.
- Adjacency relation: if \((u, v) \in E\) is an edge of \( G = (V, E) \), the vertices \( u \) and \( v \) are said to be adjacent to one another.
Sparse vs. Dense Graphs

- Let \( n \) be number of vertices, \( m \) be number of edges
- In most applications, \( n < m < O(n^2) \).
- In a **sparse** graph, \( m \) is \( O(n) \) or close to it.
- In a dense graph, \( m \) is closer to \( O(n^2) \).

Representations of Graph

- Different data structures for the representation of graphs are used in practice.
- Most common data structures are:
  - **adjacency lists**: vertices are stored as records or objects, every vertex stores a list of adjacent vertices
  - **adjacency matrix**: a two-dimensional matrix in which the rows represent source vertices and the columns represent destination vertices.
  - **unordered edge sequences**: a sequence of edges, each edge contains a pair of vertices.

**Adjacency Lists**

K: H -> T -> NULL
H: K -> B -> D -> T -> NULL
B: H -> T -> NULL
D: H -> T -> NULL
T: K -> H -> B -> D -> NULL
Adjacency Matrix

Represent $G$ by a $n \times n$ matrix $A$ where $a_{ij} = 1$ if and only if $G$ has an edge $(i, j)$

Variants:
- $a_{ij}$ = number of $(i, j)$ edges (if parallel edges)
- $a_{ij}$ = weight of $(i, j)$ edge (if any)
- $a_{ij} = 1$ if $G$ has edge $(i, j)$, $a_{ij} = -1$ if $G$ has edge $(j, i)$
Graph Algorithms

- Graph algorithms are a significant field of interest within computer science.
- Typical high-level operations associated with graphs are graph search (or graph traversal):
  - finding a path between two nodes
  - finding the shortest path from one node to another
  - finding special paths (Hamiltonian path, Euler path)

Graph Search

- Graph search refers to the problem of visiting all nodes in a graph in a particular manner.
  - a tree is a special kind of graph
  - tree traversal is a special case of graph search
- In general graph search, each node may have to be visited more than once.

Some motivations:
- check if a network is connected – can get to anywhere from anywhere else
- driving direction
- formulate a plan (example, how to fill in a Sudoku puzzle)
  - nodes = a partially completed puzzle
  - arcs = filling in one new square
- compute the “pieces” or “components” of a graph: clustering, structures of the web graph, etc.

Goals:
- find everything findable from a given vertex
- don’t explore anything twice

Generic algorithm (given graph $G$, vertex $s$)
- initially, $s$ explored, all other vertices unexplored;
- while possible:
  - choose an edge $(u, v)$ with $u$ explored and $v$ unexplored, if none halt;
  - mark $v$ explored;
Graph Search

How to choose among the possibly many "frontier" edges?
- depth-first search (using a stack – recursion)
- breadth-first search (using a queue)

Depth-First Search

Depth-first search (DFS) is an algorithm for searching/traversing a tree or a graph.
- start at root node of a tree or at some node of a graph;
- explore as far as possible along each branch before backtracking
- DFS progresses by expanding the first child node of the search tree
  - go deeper and deeper until a goal node is found or until a node that has no children is found;
  - then the search backtracks, returning to the most recent node that has not been visited

Suppose that a graph $G = (V, E)$ is represented by an adjacency matrix of size $n \times n$: $(a[i, j])_{n \times n}$.
- Let $\text{visit}(u)$ is a function which visits a node $u$.
- Let $\text{visited}[1..n]$ be a boolean array which marks whether a node $u$ has been already visited.
  - $\text{visited}[u] = 1$ if $u$ has been visited
  - $\text{visited}[u] = 0$ otherwise
- Initially, $\text{visited}[u] = 0$, for all $u = 1, \ldots, n$.

dfs(K): K, H, B, T, D.
Depth-First Search

void dfs(int u) {
    int v;
    visit(v);
    visited[u] = 1;
    for (v = 1; v < n; v++)
        if (visited[v] == 0 && a[u][v] == 1)
            dfs(v);
}

void visit(int u) {
    System.out.printf("%3d", u);
}

What is the result of dfs(0)?

Breadth-First Search

Breadth-First Search

In breadth-first search (BFS), the nodes are visited in layers:

- start at root node of a tree or at some node of a graph and visit its neighboring nodes;
- then for each of those neighbor nodes in turn, visit their neighbor nodes which were unvisited, and so on;
- All neighbor nodes are added to a queue.

BFS on a graph $G$, start vertex $s$:

- mark $s$ as visited;
- let $Q$ be a queue, initialized with $s$;
- while $Q \neq \emptyset$:
  - remove the first node of $Q$, call it $u$;
  - for each edge $(u, v)$, if $v$ is unvisited:
    - mark $v$ as visited;
    - add $v$ to $Q$;
void bfs(int s) {
    int u, v;
    visit(s);
    visited[s] = 1;
    Q.enqueue(s);
    while (!Q.isEmpty()) {
        u = Q.dequeue();
        for (v = 1; v < n; v++)
            if (visited[v] == 0 && a[u][v] == 1) {
                visit(v);
                visited[v] = 1;
                Q.enqueue(v);
            }
    }
}

What is the result of bfs(0)?

0
1
2
3
4
5

BFS can be applied to compute shortest paths on a graph.

Goal: compute \( dist(u) \), the fewest number of edges on a path from \( s \) to \( u \).

Extra code:

init:
\[
dist(u) = \begin{cases} 
0, & \text{if } u = s, \\
\infty, & \text{if } u \neq s.
\end{cases}
\]

when considering edge \( (u, v) \), if \( v \) is unvisited then set
\[
dist(v) = dist(u) + 1
\]

At termination, \( dist(u) = i \) if and only if \( u \) is in \( i \)-th layer, that is the shortest path from \( s \) to \( u \) has \( i \) edges.
Distance Measures for Social Network Graphs

- When the edges have labels, these labels might be usable as a distance measure, depending on what they represent.
- But when the edges are unlabeled, as in a “friends” graph, there is not much we can do to define a suitable distance.
- First instinct: nodes are close if they have an edge between them and distant if not.

\[ d(x, y) = \begin{cases} 
1, & \text{if there is an edge } (x, y) \\
0, & \text{otherwise} 
\end{cases} \]

- We can use two other values, for example 1 and \( \infty \), but these are not true distance measure.
  - They violate the triangle inequality when there are 3 nodes and two edges between them.

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.

Applying Standard Clustering Methods

- Two general approaches to clustering: hierarchical (agglomerative) and point-assignment.
- **What are the communities?**
  - \( \{A, B, C\} \) and \( \{D, E, F, G\} \)
  - Two subcommunities of \( \{D, E, F, G\}\): \( \{D, E, F\} \) and \( \{D, F, G\} \) with overlapping members
  - Each pair of individuals that are connected by an edge is a (uninteresting) community.
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 $\text{degree}(i)/(n - 1)$.
- 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 \( \text{indegree}(i)/(n - 1) \).
- 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
  \[
  \text{AvDist}(i) = \frac{\sum_j Dist(i, j)}{n - 1}.
  \]
- The closeness centrality is defined as \( C_C(i) = 1/\text{AvDist}(i) \).
- Note that \( 0 \leq C_C(i) \leq 1 \).

Proximity Prestige

- Defined for directed networks. Let \( \text{Influence}(i) \) be the set of nodes that can reach node \( i \) with a directed path.
- The value of \( \text{AvDist}(i) \) is computed with respect to the influence set of \( i \):
  \[
  \text{AvDist}(i) = \sum_{j \in \text{Influence}(i)} \frac{\text{Dist}(j, i)}{|	ext{Influence}(i)|}.
  \]
- The influence fraction of node \( i \) is defined as
  \[
  \text{InfluenceFraction}(i) = \frac{|	ext{Influence}(i)|}{n - 1}.
  \]
- The proximity prestige \( P_P(i) \) is defined as
  \[
  P_P(i) = \frac{\text{InfluenceFraction}(i)}{\text{AvDist}(i)}.
  \]
- 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 measure 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 $i$.

\[
f_{jk}(i) = \frac{q_{jk}(i)}{q_{jk}}
\]

- 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 betweenness 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

1. Introduction
2. Basic Concepts
3. Community Detection
   - Spectral Clustering
   - Kernighan–Lin Algorithm
   - Girvan–Newman Algorithm
   - Collective Classification
4. Influence Models
   - Population-based Model
5. Summary
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 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).

- 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.

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.

Content

1. Introduction
2. Basic Concepts
3. Community Detection
   - Spectral Clustering
     - Kernighan–Lin Algorithm
     - Girvan–Newman Algorithm
     - Collective Classification
4. Influence Models
   - Population-based Model
5. Summary
Spectral Clustering

- *A* is adjacency matrix of undirected graph *G*:
  \[ a_{ij} = \begin{cases} 
  1, & \text{if } (i, j) \text{ is an edge} \\
  0, & \text{otherwise} 
  \end{cases} \]

- *x* is a vector in \( \mathbb{R}^n \) with components \((x_1, x_2, \ldots, x_n)\).
  - Think of it as a label/value of each node of *G*.

  What is the meaning of \( A \mathbf{x} \)?

- \( j \)-th coordinate of \( A \mathbf{x} \):
  - sum of the \( x \)-values of neighbors of \( j \).
  - make this a new value at node \( j \):

\[
\begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{pmatrix}
= \lambda
\begin{pmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_n
\end{pmatrix}
\]

- That is
  \[
  A \mathbf{x} = \lambda \mathbf{x}.
  \]

Spectral Graph Theory

- Analyze the “spectrum” of matrix representing *G*.
- Spectrum: eigenvectors \( \mathbf{x}_i \) of a graph, ordered by the magnitude (strength) of their corresponding eigenvalues \( \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n \).
Suppose all nodes in $G$ have degree $d$ and $G$ is connected. What are some eigenvalues/vectors of $G$?

If $x = (1, 1, \ldots, 1)^T$ then $A x = (d, d, \ldots, d)^T = \lambda x$.

So $\lambda = d$.

We found eigenpair of $G$:

$$x = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}; \quad \lambda = d.$$

**Important properties:**
- Symmetric matrix
- Eigenvectors are real and orthogonal

---

**Matrix Representation**

**Adjacency matrix $A$**

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

What is trivial eigenpair?

- $x = (1, 1, \ldots, 1)^T$ then $L x = \vec{0}$, and so $\lambda = \lambda_1 = 0$.

**Important properties:**
- Eigenvalues are non-negative real numbers
- Eigenvectors are real and orthogonal

---

**Degree matrix $D$:** $n \times n$ diagonal matrix, $d_{ij} = $ degree of node $i$.

$$D = \begin{pmatrix} 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

**Laplacian matrix $L$:** $n \times n$ symmetric matrix, $L = D - A$.

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

---

Matrix Representation
Optimization Problem

- What is the meaning of $x^\top L x$ on $G$?
- We have

$$x^\top L x = \sum_{i,j=1}^{n} L_{ij} x_i x_j = \sum_{i,j=1}^{n} (d_{ij} - a_{ij}) x_i x_j$$

$$= \sum_{i} d_i x_i^2 - \sum_{(i,j)\in E} 2x_i x_j$$

$$= \sum_{(i,j)\in E} (x_i^2 + x_j^2 - 2x_i x_j)$$

$$= \sum_{(i,j)\in E} (x_i - x_j)^2$$

Node $i$ has degree $d_i$. So, value $x_i^2$ needs to be summed up $d_i$ times. But each edge $(i, j)$ has two endpoints so we need $x_i^2 + x_j^2$.

Spectral Clustering

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

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

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

- 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 $\mathbb{R}$.
- The values of $y_i$ can be determined by minimizing the following objective function:

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

- 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.

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

$$\Lambda = \begin{pmatrix} \lambda_{11} & 0 & \cdots & 0 \\ 0 & \lambda_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{nn} \end{pmatrix}$$

- Denote $L = \Lambda - W$. The objective function $J(y)$ can be rewritten as

$$J(y) = 2y^\top L y.$$
Spectral Clustering

- Since \( J(y) \geq 0, \forall y \), the matrix \( L \) is positive semi-definite.
- We want a non-trivial solution where \( y \equiv 0 \), therefore we need to impose a scaling constraint, such as:
  \[
  y^\top \Lambda y = 1.
  \]
- 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.

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

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

It can be shown that the optimization condition is

\[
\Lambda^{-1} L y = \lambda y,
\]
that is \( y \) is an eigenvector of \( \Lambda^{-1} L \), and the Lagrangian parameter \( \lambda \) is an eigenvalue.

There is a trivial solution: \( \lambda = 0 \) and \( y \propto (1, 1, \ldots, 1)^\top \) – Every node is mapped to the same point.

The second smallest eigenvalue is informative and provides the optimal solution.

\[
\begin{pmatrix}
3 & -1 & -1 & -1 & 0 & 0 \\
-1 & 2 & -1 & 0 & 0 & 0 \\
-1 & -1 & 3 & 0 & 0 & -1 \\
-1 & 0 & 0 & 3 & -1 & -1 \\
0 & 0 & 0 & -1 & 2 & -1 \\
0 & 0 & -1 & -1 & -1 & 3
\end{pmatrix}
\]
The second eigenvector has 3 positive and 3 negative components. This suggests that one group should be \{1, 2, 3\}, the nodes with positive components, and the other group should be \{4, 5, 6\}.

### The Spectral Clustering Algorithm

**Data:** \( W = (w_{ij})_{n \times n} \) 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. \( A \leftarrow \Lambda^{-1} \times L \);
4. compute corresponding eigenvalues \( \lambda_1, \lambda_2, \ldots, \lambda_n \) and their corresponding eigenvectors;
5. sort the eigenvalues in ascending order;
6. take top \( k \) eigenvectors \([\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_k]\) corresponding to top \( k \) smallest eigenvalues;
7. return \([\vec{v}_1, \vec{v}_2, \ldots, \vec{v}_k] \);

### Spectral Clustering – Generalization

- In the general case, we map each node to a \( k \)-dimensional embedding.
- The embedding matrix is \( Y_{n \times k} = (y_1^T, y_2^T, \ldots, y_n^T) \), where \( y_i \in \mathbb{R}^k \).
- The problem is then to minimize the trace of the \( k \times k \) matrix \( Y^TLY \) subject to the normalization constraints \( Y^T\Lambda Y = I \).
- The optimal solutions for vectors \( y_i \) can be shown to be proportional to the successive directions corresponding to the right eigenvectors of the assymmetric \( \Lambda^{-1}L \) with increasing eigenvalues.

\[
W = \begin{pmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 1 & 0
\end{pmatrix}
\]
Spectral Clustering – Example

\[ \vec{\lambda} = \begin{pmatrix} 0.00000 \\ 0.26819 \\ 1.15905 \\ 1.33333 \\ 1.33333 \\ 1.33333 \\ 1.57275 \end{pmatrix} \]

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

Exercise

The following figure is an example of a social network graph.

- Construct the Laplacian matrix for this graph.
- Find the second smallest eigenvalue and its eigenvector.
- What partition of the nodes does it suggest?
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?

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 $i$.

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.

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;
6. If no epoch with positive gain can be found then the algorithm terminates.
Kernighan–Lin Algorithm

**Data:** $W = (w_{ij})_{n \times n}$ 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 $g_t = J_{u_tv_t}$;

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 algorithm.

---

**Content**

1. Introduction
2. Basic Concepts
3. Community Detection
   - Spectral Clustering
   - Kernighan–Lin Algorithm
   - Girvan–Newman Algorithm
   - Collective Classification
4. Influence Models
   - Population-based Model
5. Summary

---

**Strength of Weak Ties**

**Edge betweenness:** Number of shortest paths passing over the edge.

![Diagram of a network with edge betweenness values](image)
Strength of Weak Ties

Edge strengths (call volume) in a real network

Edge betweenness in a real network

J. Leskovec, A. Rajaraman, J. Ullman: *Mining of Massive Datasets*

Girvan–Newman Algorithm

The Girvan–Newman algorithm is based on the intuition 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 hierarchical clustering algorithm: creates clusters by successively removing edges with the highest betweenness until the graph is disconnected into the required number of connected components.

Connected components are communities

Gives a hierarchical decomposition of the network

Because each edge removal impacts the betweenness values of some of the other edges, the betweenness values of these edges need to be recomputed after each removal.

**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 $G$
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.

We want to compute betweenness of paths starting at node $A$ of an unweighted graph $G$:

Breadth first search starting from $A$:

Count the number of shortest paths from $A$ to all other nodes:

We want to compute betweenness of paths starting at node $A$ of an unweighted graph $G$:
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.

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, $c_{ij} = 1/w_{ij}$. 

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 $j$. 

$$SP(j) = SP(i) + c_{ij},$$

where $c_{ij}$ is the length of edge $(i, j)$.\(^1\)
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 $s$.

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

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

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

$G^s$ 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 $F_{ss}(i) = 1$.

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 \( n \times (n - 1) \).

Modularity \( Q \)

Communities: sets of tightly connected nodes.

Modularity \( Q \) measures how well a network is partition into communities.

Null Model

Given real \( G \) on \( n \) nodes and \( m \) edges, we construct rewired network \( G' \):

- Same degree distribution but random connections. \( G' \) can be a multigraph.
- The expected number of edges between nodes \( i \) and \( j \) of degrees \( d_i \) and \( d_j \) equals to
  \[ \frac{d_i \times d_j}{2m} = \frac{d_i d_j}{2m} \]
- The expected number of edges in \( G' \) is
  \[ \frac{1}{2m} \sum_{i \in N} \sum_{j \in N} d_i d_j \]
  \[ = \frac{1}{2} \frac{1}{2m} \sum_{i \in N} \left( \sum_{j \in N} d_j \right) \]
  \[ = \frac{1}{4m} 2m \times 2m = m. \]
Modularity

Modularity of partitioning $S$ of graph $G$:

$$Q \propto \sum_{s \in S} \left[ \text{(expected #edges within } s) \right]$$

$$Q(S, G) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in S} \sum_{j \in S} \left( A_{ij} - \frac{d_id_j}{2m} \right).$$

- We use $1/(2m)$ to normalize cost so that $-1 \leq Q \leq 1$.
- $Q \in [0.3, 0.7]$ means significant community structure.

Modularity is useful for selecting the number of clusters.

Exercise

The following figure is an example of a social network graph.

Use the Girvan–Newman approach to find the number of shortest paths from each of the following nodes that pass through each of the edges:

- $A$
- $B$

Content

1. Introduction
2. Basic Concepts
3. Community Detection
   - Spectral Clustering
   - Kernighan–Lin Algorithm
   - Girvan–Newman Algorithm
   - Collective Classification
4. Influence Models
   - Population-based Model
5. Summary
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 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 $A$, such as Naive Bayes classifier to compute the likelihood that it belongs to a particular class:

$$P(y_i | \{\text{content features, link features}\})$$

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 $A$

1 for $t \leftarrow 1$ to $T$ do
2 extract link features at each node with current training data;
3 train classifier $A$;
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

Infection: Population-based Model

- Assume that each node can interact with any other node: this model does not depend on the topology of the network.
- Many variants of the infection model differ from each other in terms of the kind of state transitions that are allowed.
- Two-state and three-state models, frequently used in modeling the spread of infectious disease.
Infection: Population-based Model

- \( S \): susceptible population, \( I \): infected pop., \( R \): recovered pop.

Each model can be mathematically described through a system of differential equations.

\[
\begin{align*}
S & \rightarrow \beta I \\
S & \rightarrow I \\
S & \rightarrow I \\
S & \rightarrow I \\
S & \rightarrow I \\
\end{align*}
\]

\( \beta, \gamma \) are rates of the transitions: how many switch per unit of time.

We use \( S(t), I(t), \) and \( R(t) \) to represent the proportions of the population in that state at time \( t \).

The initial conditions at time 0 are \( S(0), I(0), \) and \( R(0) \).

The SI model is described by the following pair of differential equations:

\[
\begin{align*}
\frac{dS(t)}{dt} &= -\beta S(t)I(t), \\
\frac{dI(t)}{dt} &= \beta S(t)I(t).
\end{align*}
\]

The transition rates are proportional to the product \( S(t)I(t) \) at each time \( t \).

When \( t \) is small, \( I(t) \)'s growth is similar to exponential growth.

When \( t \) becomes large, the ratio \( I(t) \) approaches 1.

\[
\begin{align*}
\text{Population evolution in the SI model}
\end{align*}
\]
Infection: SIS Model

- The SIS model assumes that once infected, a person stays infected forever.
- In some diseases, a person can become non-infected but still remain susceptible to further infections.
- The **SIS model** is described by the following equations:

\[
\frac{dS(t)}{dt} = -\beta S(t)I(t) + \gamma I(t),
\]
\[
\frac{dI(t)}{dt} = \beta S(t)I(t) - \gamma I(t).
\]

Infection: SIS Model

- We can guess that if \( \beta < \gamma \), the infected proportion depletes exponentially.
- If \( \beta > \gamma \), we will see a sigmoidal curve of \( I(t) \) going up, but not 100% since some of the infected will be going back to the susceptible state.
- The exact saturation percentage of \( I(t) \) depends on \( \beta/\gamma \).

Again, using \( S(t) = 1 - I(t) \) and solving the resulting differential equation in \( I(t) \), we have

\[
I(t) = (1 - \gamma/\beta) \frac{ce^{(\beta-\gamma)t}}{1 + ce^{(\beta-\gamma)t}},
\]

for some constant \( c \) that depends on the initial condition.

At equilibrium, some people are not infected. The important constant

\[ \sigma = \beta/\gamma, \]

is called the **basic reproduction number**.
Both SI and SIS models miss a common feature in many diseases: once infected and then recovered, a person becomes immunized. This is the $R$ state.

In the SIR model, the infected population eventually goes down to 0. The dynamics are described by the following equations:

$$\frac{dS(t)}{dt} = -\beta S(t)I(t),$$
$$\frac{dI(t)}{dt} = \beta S(t)I(t) - \gamma I(t),$$
$$\frac{dR(t)}{dt} = \gamma I(t).$$

Here, $\sigma = \beta/\gamma$ is the contact rate $\beta$ times the average infection period $1/\gamma$.

There is no closed-form solution to these differential equations. The trajectory of this SIR model has the following properties over a period of time $[0, T]$.

- If $\sigma S(0) \leq 1$ then $I(t)$ decreases to 0 as $t \to \infty$. The initial value $S(0)$ is not large enough to create an epidemic.
- If $\sigma S(0) > 1$ then $I(t)$ increases to a maximum of

$$I_{\text{max}} = I(0) + S(0) - 1/\sigma - \log(\sigma S(0))/\sigma,$$

then decreases to 0 as $t \to \infty$. This is the typical curve of an epidemic outbreak.

$S(t)$ is always a decreasing function. Eventually, everyone is recovered.

### Content

1. **Introduction**
2. **Basic Concepts**
3. **Community Detection**
   - Spectral Clustering
   - Kernighan–Lin Algorithm
   - Girvan–Newman Algorithm
   - Collective Classification
4. **Influence Models**
   - Population-based Model
5. **Summary**

### Summary

- Some basic concepts in social network analysis
- Challenges in community detection
- Some typical algorithms for community detection:
  - Spectral clustering
  - Kernighan–Lin algorithm
  - Girvan-Newman algorithm
  - Iterative classification algorithm
- Infection models for influence analysis