Community Detection in Social Network

WANG JIANWEI & YAO KAI
Outlines

- Introduction
- Methods of community detection
  - Traditional methods:
    - Graph partitioning
    - Hierarchical clustering
    - Partitional clustering
  - Divisive algorithms
  - Dynamic algorithms
  - Label propagation algorithm
- Some CD-related problems
  - Overlapping
  - Hubs and outliers
  - Content-based
  - Local communities
Introduction

- Graphs representing real systems are not regular like, e.g., lattices.
- In a random graph, the distribution of edges among the vertices is highly homogeneous.
- Real networks are not random graphs, as they display big inhomogeneities, revealing a high level of order and organization.
  - Many vertices with low degree coexist with some vertices with large degree.
  - The distribution of edges is not only globally, but also locally inhomogeneous, with high concentrations of edges within special groups of vertices, and low concentrations between these groups.
Communities, also called clusters or modules, are groups of vertices which probably share common properties and/or play similar roles within the graph.

No quantitative definition of community is universally accepted. As a matter of fact, the definition often depends on the specific system at hand and/or application one has in mind.

There must be more edges “inside” the community than edges linking vertices of the community with the rest of the graph.

Moreover, in most cases, communities are algorithmically defined.

We distinguish three classes of definitions: local, global and based on vertex similarity.
Examples of social networks (1)

Fig. 1: Zachary’s network of karate club members

34 vertices → 34 members of this club

Edges connect individuals who were observed to interact outside the activities of the club.

At some point, a conflict between the club president and the instructor led to the fission of the club in two separate groups, supporting the instructor and the president, respectively (indicated by squares and circles).
Examples of social networks (2)

Fig. 2: The system of the World Wide Web

Vertices $\rightarrow$ web pages

the hyperlinks that make users move from one page to another as edges

Hyperlinks are directed
Community can have concrete applications

- **Clustering Web clients** who have similar interests and are geographically near to each other may improve the performance of services provided on the World Wide Web, in that each cluster of clients could be served by a dedicated mirror server.

- **Identifying clusters of customers** with similar interests in the network of purchase relationships between customers and products of online retailers enables to set up efficient recommendation systems, that better guide customers through the list of items of the retailer and enhance the business opportunities.

- **Clusters of large graphs** can be used to create data structures in order to efficiently store the graph data and to handle navigational queries, like path searches.

...
The aim of community detection in graphs is to identify the modules and, possibly, their hierarchical organization, by only using the information encoded in the graph topology.

The first analysis of community structure was carried out by Weiss and Jacobson (Weiss and Jacobson, 1955), who searched for work groups within a government agency.

Identifying graph communities is a popular topic in computer science, too.
The problem of graph partitioning consists in dividing the vertices in $g$ groups of predefined size, such that the number of edges lying between the groups is minimal.

The number of edges running between clusters is called cut size.

Specifying the number of clusters of the partition is necessary.

Specifying the size is also necessary.

Graph partitioning is a fundamental issue in parallel computing, circuit partitioning and layout, and in the design of many serial algorithms, including techniques to solve partial differential equations and sparse linear systems of equations.

Most variants of the graph partitioning problem are NP-hard.

Kernighan-Lin algorithm, spectral bisection method, level-structure partitioning, geometric algorithm, multilevel algorithm etc.
Kernighan-Lin algorithm (1)

- The authors were motivated by the problem of partitioning electronic circuits onto boards: the nodes contained in different boards need to be linked to each other with the least number of connections.

- The procedure is an optimization of a benefit function $Q$, which represents the difference between the number of edges inside the modules and the number of edges lying between them.

- Procedures:
  - An initial partition of the graph in two clusters of the predefined size.
  - Subsets consisting of equal numbers of vertices are swapped between the two groups, so that $Q$ has the maximal increase.
  - After a series of swaps with positive and negative gains, the partition with the largest value of $Q$ is selected and used as starting point of a new series of iterations.
Kernighan-Lin algorithm (2)

- **Advantages:**
  - Quite fast, scaling as $O(n^2 \log n)$ (n being as usual the number of vertices), if only a constant number of swaps are performed at each iteration.

- **Disadvantages:**
  - The partitions found by the procedure are strongly dependent on the initial configuration and other algorithms can do better. It is preferable to start with a good guess about the sought partition, otherwise the results are quite poor.
  - The most expensive part is the identification of the subsets to swap, which requires the computation of the gains/losses for any pair of candidate subsets.
  - It can be extended to extract partitions in any number of parts, however the run-time and storage costs increase rapidly with the number of clusters.
It is based on the properties of the spectrum of the Laplacian matrix.

Every partition of a graph with \( n \) vertices in two groups can be represented by an index vector \( s \), whose component \( s_i \) is \(+1\) if vertex \( i \) is in one group and \(-1\) if it is in the other group.

The cut size \( R \) of the partition of the graph in the two groups can be written as

\[
R = \frac{1}{4} s^T L s
\]

\[
s = \sum_i a_i v_i
\]

Minimizing \( R \) equals to the minimization of the sum on the right-hand side of the rightmost equation.

If the second lowest eigenvector \( \lambda_2 \) is close enough to zero, a good approximation of the minimum can be attained by choosing \( s \) parallel to the corresponding eigenvector \( v_2 \). But, the index vector cannot be perfectly parallel to \( v_2 \) by construction.
The best choice is to match the signs of the components. One can set $s_i = +1$ ($-1$) if $v^2_i > 0$ ($< 0$).

In this case, if one aims at a split in $n_1$ and $n_2 = n - n_1$ vertices, the best strategy is to order the components of the Fiedler vector $v_2$ from the lowest to the largest values and to put in one group the vertices corresponding to the first $n_1$ components from the top or the bottom, and the remaining vertices in the second group.

This procedure yields two partitions: the better solution is naturally the one that gives the smaller cut size.

Advantages:

- The spectral bisection method is quite fast. ($O(n^3)$).
- The method gives in general good partitions, that can be further improved by applying the Kernighan-Lin algorithm.
Other methods of Graph Partitioning

- Level-structure partitioning
- Geometric algorithm
- Multilevel algorithm
- Graphs can be also partitioned by minimizing measures that are affine to the cut size, like conductance, ratio cut and the normalized cut.

\[
\Phi(C) = \frac{c(C, G \setminus C)}{\min(k_C, k_{G \setminus C})} \quad \Phi_C(C) = \frac{c(C, G \setminus C)}{n_C n_{G \setminus C}} \quad \Phi_N(C) = \frac{c(C, G \setminus C)}{k_C}
\]
Hierarchical clustering

- **Hierarchical clustering**, clustering techniques that reveal the **multilevel structure** of the graph, is very common in social network analysis, biology, engineering, marketing, etc.

- Hierarchical clustering techniques **aim** at **identifying groups of vertices** with high similarity, and can be classified in two categories:
  - **Agglomerative algorithms**, in which clusters are iteratively **merged** if their similarity is sufficiently high;
  - **Divisive algorithms**, in which clusters are iteratively **split** by removing edges connecting vertices with low similarity.

- **Procedures:**
  - The starting point of any hierarchical clustering method is the definition of a similarity measure between vertices.
  - After a measure is chosen, one computes the similarity for each pair of vertices, no matter if they are connected or not.
  - At the end of this process, one is left with a new $n \times n$ matrix $X$, the similarity matrix

- The procedure can be better illustrated by means of **dendrograms**.
Several possible definitions of similarity

- Single Linkage
  - Complete Linkage
  - Group Average Linkage
  - Centroid Linkage
  - Median Linkage
Single Linkage clustering

- Also, known as the nearest neighbor technique.
- Similarity between groups is defined as that of the closest pair of data, where only pairs consisting of one record from each group are considered.
Several possible definitions of similarity

- Single Linkage
- Complete Linkage
- Group Average Linkage
- Centroid Linkage
- Median Linkage
The similarity between two clusters is given by the distance between their **most distant members**.
Several possible definitions of similarity

- Single Linkage
- Complete Linkage
- Group Average Linkage
- Centroid Linkage
- Median Linkage
The similarity between two clusters is defined as the average of the similarity between all pairs of records (one from each cluster).

\[ d_{AB} = \frac{1}{6} (d_{13} + d_{14} + d_{15} + d_{23} + d_{24} + d_{25}) \]
Several possible definitions of similarity

- Single Linkage
- Complete Linkage
- Group Average Linkage
- Centroid Linkage
- Median Linkage
The similarity between two clusters is defined as the similarity between the mean vectors of the two clusters.

\[ d_{AB} = d_{ab} \]

where \( a \) is the mean vector of the cluster A and \( b \) is the mean vector of the cluster B.
Several possible definitions of similarity

- Single Linkage
- Complete Linkage
- Group Average Linkage
- Centroid Linkage
- Median Linkage
Disadvantage of the Centroid Clustering: When a large cluster is merged with a small one, the centroid of the combined cluster would be close to the large one, i.e. The characteristic properties of the small one are lost.

After we have combined two groups, the mid-point of the original two cluster centers is used as the center of the newly combined group.
Hierarchical clustering

- **Advantages:**
  - It does not require a preliminary knowledge on the number and size of the clusters.

- **Disadvantages:**
  - It does not provide a way to discriminate between the many partitions obtained by the procedure, and to choose that or those that better represent the community structure of the graph.
  - The results of the method depend on the specific similarity measure adopted.
  - The procedure also yields a hierarchical structure by construction, which is rather artificial in most cases, since the graph at hand may not have a hierarchical structure at all.
  - Vertices of a community may not be correctly classified, and in many cases some vertices are missed even if they have a central role in their clusters.
  - Vertices with just one neighbor are often classified as separated clusters, which in most cases does not make sense.
  - Agglomerative hierarchical clustering does not scale well.
Partitional clustering

- The number of clusters is preassigned, say k.
- The points are embedded in a metric space, so that each vertex is a point and a distance measure is defined between pairs of points in the space.
- The distance is a measure of dissimilarity between vertices.
- The goal is to separate the points in k clusters such to maximize/minimize a given cost function based on distances between points and/or from points to centroids, i.e. suitably defined positions in space.
Some of the most used functions

- **Minimum k-clustering:** The cost function here is the **diameter** of a cluster, which is the **largest** distance between two points of a cluster. The points are classified such that the largest of the k cluster diameters is the smallest possible.

- **k-clustering sum:** Same as minimum k-clustering, but the diameter is replaced by the **average distance** between all pairs of points of a cluster.

- **k-center:** For each cluster i, one defines a reference point $x_i$, the centroid, and computes the **maximum** $d_i$ of the distances of each cluster point from the centroid. The clusters and centroids are self-consistently chosen such to minimize the largest value of $d_i$.

- **k-median:** Same as k-center, but the maximum distance from the centroid is replaced by the **average distance**.
Cost function (Squared error function):

$$\sum_{i=1}^{k} \sum_{x_j \in S_i} \|x_j - c_i\|^2$$

where $S_i$ indicates the subset of points of the $i$-th cluster and $c_i$ its centroid.

- The k-means problem can be simply solved with the Lloyd’s algorithm.
  - One starts from an initial distribution of centroids such that they are as far as possible from each other.
  - In the first iteration, each vertex is assigned to the nearest centroid.
  - Next, the centers of mass of the $k$ clusters are estimated and become a new set of centroids, which allows for a new classification of the vertices, and so on.
  - After a small number of iterations, the positions of the centroids are stable, and the clusters do not change any more.
Fuzzy k-means clustering(1)

- **Cost function:**

\[
J_m = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^m \|x_i - c_j\|^2
\]

where \(u_{ij}\) is the membership matrix, which measures the degree of membership of point \(i\) (with position \(x_i\)) in cluster \(j\), \(m\) is a real number greater than 1 and \(c_j\) is the center of cluster \(j\).

- The matrix \(u_{ij}\) is normalized and the membership \(u_{ij}\) is related to the distance of point \(i\) from the center of cluster \(j\),

\[
u_{ij} = \frac{1}{\sum_{l=1}^{k} \left( \frac{\|x_i - c_l\|^2}{\|x_i - c_j\|^2} \right)^{m-1}}
\]
The cost function $J_m$ can be minimized by iterating equations (1) and (2).

One starts from some initial guess for $u_{ij}$ and uses Eq. (1) to compute the centers, which are then plugged back into Eq. (2) and so on.

The process stops when the corresponding elements of the membership matrix in consecutive iterations differ from each other by less than a predefined tolerance.

It can be shown that this procedure indeed delivers a local minimum of the cost function $J_m$.

\[
c_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m} \tag{1}
\]

\[
u_{ij} = \frac{1}{\sum_{l=1}^{k} \left( \frac{\|x_i - c_l\|}{\|x_i - c_l\|} \right)^{m-1}} \tag{2}
\]
Partitional clustering

- **Disadvantages:**
  - The number of clusters must be specified at the beginning, the method is not able to derive it.
  - The embedding in a metric space can be natural for some graphs, but rather artificial for others.
• Introduction
• Methods of community detection
  • Traditional methods
    • Graph partitioning
    • Hierarchical clustering
    • Partitional clustering
  • Divisive algorithm
  • Dynamic algorithm
  • Label propagation algorithm
• Some CD-related problems
  • Overlapping
  • Hubs and outliers
  • Content based
  • Local communities
2.2 Divisive algorithm

• Main idea:
  • Detect the edges that connect vertices of different communities and remove them

• How to find intercommunity edges?
  • Edge betweenness
  • Edge clustering coefficient

• Edge betweenness
  • find the **shortest paths** between all pairs of vertices and count how many run along each edge

Case 1: Whenever there is a shortest path go through an edge, then the value of this edge will increase 1

Case 2: multiple shortest paths between a pair of vertices are given equal weights summing to 1

\[
\frac{11}{6} = 1 + \frac{1}{2} + \frac{1}{3}
\]
• Algorithm:
  1. Calculate betweenness scores for all edges
  2. Remove the edge with highest score
  3. Recalculate for all remaining edges
  4. Repeat from step 2

• How do we know when the communities found by algorithm are good ones?
  • Modularity
    \[ Q = \sum_i (e_{ii} - a_i^2) = \text{Tr e} - \| e^2 \| \]
  • Calculate \( Q \) for each split of a network into communities
  • Look for local peaks

• Computationally expensive
  • It requires repeated evaluation
  • The betweenness depends on whole graph
Edge clustering coefficient

- The number of triangles to which a given edge belongs

\[ \tilde{C}_{i,j}^{(3)} = \frac{z_{i,j}^{(3)} + 1}{\min[(k_i - 1), (k_j - 1)\]} \]

- Intuitive idea:
  - edges connecting nodes in different communities are included in few or no triangles
  - Edges within communities tend to form many triangles

- Efficient
  - Running time: \( O(n^2) \)
2.3 Dynamic algorithm

• Main idea:
  • Random walk can define a distance between pairs of vertices
  • random walks on a graph tend to get “trapped” into densely connected parts corresponding to communities

• Adjacency matrix $A$:
  • $A_{ij} = 1$ if $(v_i, v_j) \in E$
  • $A_{ij} = 0$ otherwise

• Transition matrix $P$:
  • $P_{ij} = \frac{A_{ij}}{d(i)}$

\[
\begin{array}{cccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
 v_1 & 1 & 1 & 1 & 0 & 0 & 0 \\
v_2 & 1 & 1 & 1 & 0 & 0 & 0 \\
v_3 & 1 & 1 & 1 & 0 & 0 & 0 \\
v_4 & 0 & 0 & 1 & 1 & 1 & 1 \\
v_5 & 0 & 0 & 0 & 1 & 1 & 1 \\
v_6 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
 v_1 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
v_2 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\
v_3 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & 0 & 0 \\
v_4 & 0 & 0 & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\
v_5 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
v_6 & 0 & 0 & 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\
\end{array}
\]

\[
\begin{array}{cccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
 v_1 & \frac{11}{36} & \frac{11}{36} & \frac{11}{36} & \frac{3}{36} & 0 & 0 \\
v_2 & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & 0 \\
v_3 & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} \\
v_4 & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} \\
v_5 & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} & \frac{48}{36} \\
v_6 & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} & \frac{36}{36} \\
\end{array}
\]

\[
\begin{array}{cccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
 v_1 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_2 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_3 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_4 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_5 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_6 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
\end{array}
\]

\[
\begin{array}{cccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
 v_1 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_2 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_3 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_4 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_5 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_6 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
\end{array}
\]

\[
\begin{array}{cccccc}
  & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \\
 v_1 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_2 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_3 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_4 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_5 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
v_6 & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} & \frac{0}{36} \\
\end{array}
\]

$(P^t)_{ij}$: The probability of going from $i$ to $j$ through a random walk of length $t$
How to define a distance between vertices

1. If vertices $i$ and $j$ in the same community, $P_{ij}^t$ should be high.
2. If vertices $i$ and $j$ in the same community, $P_{ik}^t \approx P_{jk}^t$ for all $k \in G$.
3. $P_{ij}^t$ is influenced by the degree $d(j)$

- Distance between vertices:
  \[
  r_{ij} = \sqrt{\sum_{k=1}^{n} \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}} = \left\| D^{-\frac{1}{2}}P_i^t - D^{-\frac{1}{2}}P_j^t \right\|
  \]

- Distance between a vertex and a community:
  \[
  P_{C,j}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t
  \]

- Distance between two communities:
  \[
  r_{C_1C_2} = \left\| D^{-\frac{1}{2}}P_{C_1}^t - D^{-\frac{1}{2}}P_{C_2}^t \right\| = \sqrt{\sum_{k=1}^{n} \frac{(P_{C_1k}^t - P_{C_2k}^t)^2}{d(k)}}
  \]
• Algorithm:
  1. Start from $n$ communities
  2. Choose two communities $C_1$ and $C_2$ with the smallest distance
  3. Merge $C_1$ and $C_2$, get $C_3$
  4. Update the distance between communities
2.4 Label propagation algorithm

- **Main idea:**
  - Node $x$ determines its community based on the communities of its neighbors.

- **Algorithm:**
  1. Every node is initialized with a unique label.
  2. At every step each node adopts the label that most of its neighbors currently have.
  3. Repeat step 2, until converge.

- In the end, nodes having the same labels are grouped together as communities.
3 Some CD-related problems

3.1 Detection of overlapping communities

• **Motivation:** Most of the actual networks are made of highly overlapping cohesive groups of nodes

• Clique Percolation Method (CPM)
  • Main idea:
    • The internal edges of a community are likely to form *cliques* due to their high density
  • k-clique
    • a complete graph with *k* vertices
  • Adjacent k-cliques
    • Two k-cliques are *adjacent* if they share *k*-1 vertices
  • k-clique community
    • The largest connected subgraph obtained by all adjacent k-cliques
• Algorithm
  1. Search for maximal cliques
  2. Build a clique-clique overlap matrix $O = n_c \times n_c$
  3. Keep the entries of $O$ which are larger than $k - 1$
  4. Find the connected components of the resulting matrix

• Cons:
  • It assumes the graph has a large number of cliques
  • Many “leaves”
  • It is hard to choose a proper $k$

• Other methods
  • Locally optimal clusters
  • Vertex similarity
  • Split a node in multiple parts
3.2 Detection of communities with hubs and outliers

- **Motivation**: Hubs and outliers are commonly seen in social networks and play an important role in various applications.

- **Structural Clustering**:
  - Structural similarity
    - \( \sigma(v, w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)||\Gamma(w)|}} \)
  - \( \varepsilon \) _neighborhood_
    - \( N_\varepsilon(v) = \{w \in \Gamma(v) | \sigma(v, w) \geq \varepsilon\} \)
  - Core node
    - its \( \varepsilon \) _neighborhood_ is larger than \( \mu \)
• Algorithm:
  1. All vertices are labeled as unclassified
  2. For each unclassified vertex, check if it is a core
     • YES: create a new cluster, expand this cluster
     • NO: it is labeled as non-member
  3. Repeat step 2 until there is no unclassified vertex
  4. For each non-member vertex
     • If it connects two or more different clusters, label it as hub
     • Else, label is as outlier

• Complexity
  • The average running cost: $O\left(\frac{|E|^2}{|V|}\right)$
  • The worst running cost: $O\left(|V|^3\right)$
  • Costly, because it evaluates the density for all adjacent node
  • SCANN++: more efficient [2015 VLDB]
3.3 Detection of content-based communities

- **Motivation:**
  - incorporate both the network structure and content to improve community detection performance.

- **Structural/Attribute clustering**
  - **Problem definition:** Given an attributed graph $G$, partition it into $k$ disjoint subgraphs. A desired clustering should be:
    1. Vertices within one community are close to each other in terms of structure
    2. Vertices within one community have similar attribute value
• Neighborhood Random Walk Distance:
\[ d(v_i, v_j) = \sum_{\tau: \text{length}(\tau) \leq l} p(\tau)c(1-c)^{\text{length}(\tau)} \]

• Neighborhood Random Walk Distance Matrix:
\[ R^l = \sum_{\gamma=1}^{l} c(1-c)^\gamma P^\gamma \]

• Attribute Augmented Graph
  • Given an attributed graph \( G = (V, E, \Lambda) \) where \( \Lambda = \{a_1, ..., a_m\} \). The domain (the set of possible values) of attribute \( a_i \) is \( \text{Dom}(a_i) = \{a_{i1}, ..., a_{in_i}\} \) where \( |\text{Dom}(a_i)| = n_i \).
  • An attribute augmented graph is denoted as \( G_a = (V \cup V_a, E \cup E_a) \)
  • Add attribute vertices: \( V_a = \{V_{ij}\}_{i=1, j=1}^{m, n_i} \)
  • An edge \((v_i, v_{jk}) \in E_a \) iff \( a_j(v_i) = a_{jk} \)

\( c \): restart probability
\( p(\tau) \): transition probability
\( P \): transition probability matrix
• With the attribute edges, authors who are originally isolated become much closer if they share a common topic.

• The more attribute values two vertices share, the more random walk paths exist between the pair of vertices.

• K-means clustering method, with the random walk distance as a pairwise similarity measure.

• Cons:
  • The creation of attribute augmented graph is costly
  • $|V_a| = m \times \sum_{i=1}^{m} n_i$
  • $|E_a| = |V| \times m$
3.4 Detection of local communities

**Motivation**
- In many application scenarios, we are interested in discovering the community defined by a given set of nodes. (community search)
- The goal is to find the best community containing those nodes.

**Community search based on minimum degree**
- **Problem Definition**: Given an undirected graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, and a goodness function $f$, we seek to find an induced subgraph $H = (V_H, E_H)$ of $G$, such that
  1. $V_H$ contains $Q$ ($Q \subseteq V_H$);
  2. $H$ is connected
  3. $f(H)$ is maximized among all feasible choices for $H$

How to choose $f$?
1. The average degree $f_a(H)$ of the nodes in $H$
2. The minimum degree $f_m(H)$ of the nodes in $H$
Algorithm

1. We start by setting $G_0 = G$, and we proceed by deleting one node in each step.
2. At the $t$-th step, we consider a node $u$ that has minimum degree in $G_{t-1}$.
3. We get graph $G_t$ by deleting $u$ and all edges incident to $u$ from $G_{t-1}$.
4. Algorithm terminates if either:
   - At least one of the query node $Q$ has the minimum degree.
   - The query nodes $Q$ are no longer connected.

Complexity

- Make a list of nodes with degree $d$. (one separate list for each degree)
- $O(n + m)$
References

1. Fortunato S. Community detection in graphs [J].
5. Raghavan U N, Albert R, Kumara S. Near linear time algorithm to detect community structures in large-scale networks [J].
11. Sozio M, Gionis A. The community-search problem and how to plan a successful cocktail party [C].