Community Structure Identification

Maths Foundation for Computer Science (CO5263)

HCMC University of Technology

Department of Computer Science and Engineering

March 2024

Professor: Nguyễn An Khương

Members:

Lê Hoàng Anh Tài	2370761
Nguyễn Duy Hùng	2370563
Trịnh Hoàng Xuân Hãn	2370562
Lưu Nguyễn Bảo Ngọc	2370565
Nguyễn Văn Hoàng Khang	2110242
Nguyễn Văn Khoa	2370499

Agenda

Introduction

Community Structure Identification is a fundamental concept in network analysis that seeks to reveal the inherent organization of complex networks by pinpointing clusters of nodes that exhibit strong internal connectivity while being sparsely connected to nodes in other clusters.

Use case: Customer Network

Suppose that we are running an online bookstore. Customers may be interested in more than one type of book. They can find their interests in the latest fantasy thriller by Stephen King while sharing some of their pleasures for mathematical textbooks. Finding which types of books each customer invests in can help us improve our service through investigations in the detected communities. In this case, each book is represented by a vertex in a graph and an edge between two books is created if these two are bought by the same customer.

Introduction

Algorithm: The Louvain algorithm

- The Louvain method is an algorithm to detect communities in large networks. It maximizes a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.
- The Louvain algorithm is a hierarchical clustering algorithm, that recursively merges communities into a single node and executes the modularity clustering on the condensed graphs.

Data set: Political books

- The dataset represents interactions between political books.
- Each book is treated as a node in the network.
- Edges between books indicate frequent co-purchasing by the same buyers, suggesting common interests or topics.

Preliminaries

Modelize a social network

- We define a graph $G = (V, E)$, where V is set of vertices, or nodes and E is set of edges. If there exists an edge between two vertices u and v, we say that u and v have a relationship and their edge is denoted as $\{u, v\}$.
- We can modelize a social network as a graph by assigning entities to vertices and relationships between them to edges.
- Use adjacency matrix $|V| \times |V|$ to represent a graph.

Preliminaries

Metric for dectecting community

Given a subgraph C of G, $|C| = n_C$, $|G| = n$. Then we denote:

- Internal degree k_v^{int} for $v \in C$.
- External degree k_v^{ext} for $v \in C$.
- Number of internal edge E_{C-C} of subgraph C.
- Number of external edge E_C of subgraph$C$.

Preliminaries

Metric for dectecting community

- **Intro-cluster density of subgraph C, denoted as** $\delta_{int}(C)$

$$\delta_{int}(C) = \frac{|E_{C-C}|}{n_C(n_C - 1)} \tag{1}$$

- **Inter-cluster density of subgraph C, denoted as** $\delta_{ext}(C)$:

$$\delta_{ext}(C) = \frac{|E_C| - |E_{C-C}|}{n_C(n - n_C)} \tag{2}$$

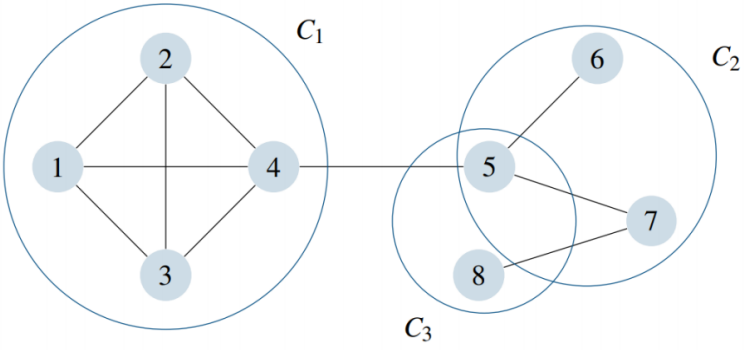
- **Average density of a graph, denoted as** $\delta(G)$:

$$\delta(G) = \frac{|E|}{n(n-1)/2} \tag{3}$$

- If C is a community in the graph, we expect $\delta_{int}(C) > \delta(G)$ and $\delta_{ext}(C) < \delta(G)$.
- **Target:** Maximize $\delta_{int}(C) - \delta_{ext}(C)$

Preliminaries

Example 1: Given a graph G = (V,E) as figure 1 below with set of vertices V = $\{1, 2, 3, 4, 5, 6, 7, 8\}$ and set of edges E = $\{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{2, 4\}, \{1, 4\}, \{4, 5\}, \{5, 6\}, \{5, 7\}, \{7, 8\}\}$. Examine three subgraphs C_1, C_2, C_3 as figure below:



Preliminaries

We estimate all metrics:

- Average density of graph G:

$$\delta(G) = \frac{|E|}{n(n-1)/2} = \frac{10}{8(8-1)/2} = \frac{5}{14} \approx 0.357$$

- Intra-cluster density of subgraph C_1:

$$\delta_{int}(C_1) = \frac{|E_{C_1-c_1}|}{n_{C_1}(n_{C_1}-1)/2} = \frac{6}{4(4-1)/2} = 1$$

- Intra-cluster density of subgraph C_2:

$$\delta_{int}(C_2) = \frac{|E_{C_2-c_2}|}{n_{C_2}(n_{C_2}-1)/2} = \frac{2}{3(3-1)/2} \approx 0.667$$

- Intra-cluster density of subgraph C_3:

$$\delta_{int}(C_3) = \frac{|E_{C_3-c_3}|}{n_{C_3}(n_{C_3}-1)/2} = \frac{0}{2(2-1)/2} = 0$$

Preliminaries

- Inter-cluster density of subgraph C_1:

$$\delta_{ext}(C_1) = \frac{|E_{C_1}| - |E_{C_1 - C_1}|}{n_{C_1}(n - n_{C_1})} = \frac{7 - 6}{4(8 - 4)} = \frac{1}{16} \approx 0.063$$

- Inter-cluster density of subgraph C_2:

$$\delta_{ext}(C_2) = \frac{|E_{C_2}| - |E_{C_2 - C_2}|}{n_{C_2}(n - n_{C_2})} = \frac{4 - 2}{3(8 - 3)} = \frac{2}{15} \approx 0.133$$

- Inter-cluster density of subgraph C_3:

$$\delta_{ext}(C_3) = \frac{|E_{C_3}| - |E_{C_3 - C_3}|}{n_{C_3}(n - n_{C_3})} = \frac{4 - 0}{2(8 - 2)} = \frac{1}{3} \approx 0.333$$

Based on the values that we just estimate, we have some reviews:

- $\delta_{int}(C_1) = 1 > \delta(G)$ and $\delta_{ext}(C_1) = 0.063 < \delta(G)$, so C_1 is likely a community.
- $\delta_{int}(C_2) = 0.667 > \delta(G)$ and $\delta_{ext}(C_2) = 0.133 < \delta(G)$, so C_2 is likely a community.
- $\delta_{int}(C_3) = 0 < \delta(G)$ and $\delta_{ext}(C_3) = 0.333 < \delta(G)$, so C_3 is not likely a community.

Preliminaries

Communication categorization

An entity (or a node) can be a member of more than one community. This leads to variety in community structure. We divide the community structure into three types:

- **Non-overlapping community**: refers to a community that is separate from others, where each member is exclusive to that community and does not participate in any other community.
- **An overlapping community**: is a type of community where a member can also be part of another community.
- **A nested community:** is a type of community that exists as a subset within a larger community.

Preliminaries

Local graph and identifying community on local graph

A Local Graph is a smaller part of a larger graph, focusing on the relationships between one or a few vertices. In the context of social networks, a local graph is a subgraph where the set of vertices includes a central node, and other nodes, referred to as neighbor nodes, are connected to it. This local graph is utilized to depict the relationships of the central node.

The local graph has several characteristics:

- The central node is connected to other vertices by edges.
- Many vertices have a degree of 1, indicating that these vertices are connected only to the central node and no other vertices.
- All edges in the local graph hold significance to the central node.

In a local graph, communities exhibit little overlap, and the number of vertices within a community can vary.

Preliminaries

Modularity Measure: : Computing the distribution of edges inside a community compared to a random graph with same edges' distribution

$$Q = \frac{1}{2m} \sum_{ij} (A_{ij} - P_{ij}) \delta(C_i, C_j) \tag{4}$$

Where:

- A_{ij} is entry of row *ith* and column *jth* of adjacency matrix A of the graph.
- m is the number of edges
- P_{ij} is the expectation of number of edge between i and j
- δ function has the value 1 if i,j are in the same community, and 0 otherwise.

Preliminaries

Another formula for Modularity Measure

$$Q = \sum_{C} \left[\frac{l_C}{2m} - \left(\frac{d_C}{2m} \right)^2 \right]$$

Where:

- C is a community.
- l_c is number of edges in community C.
- d_c is total degree of vertices in C.

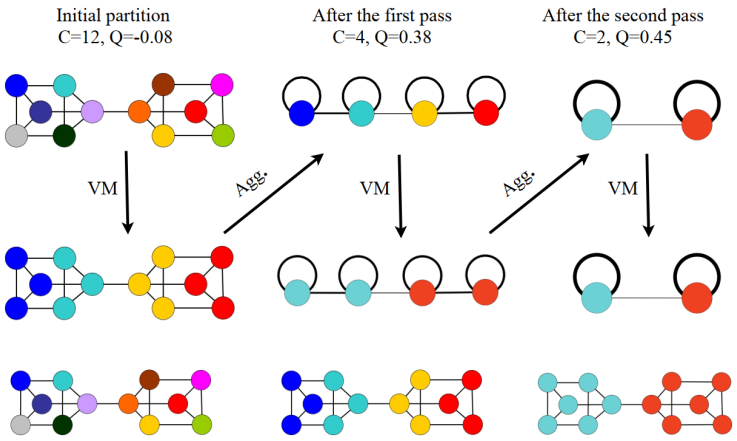
The Louvain algorithm

1. Algorithm description

The Louvain method is a greedy optimization heuristic originally proposed to optimize modularity, but its flexibility allows to optimize other quality functions as well, as described below. The approach consists in two phases that are iteratively repeated, until a maximum of modularity is obtained.

- **First phase: vertex mover (VM).** The first phase begins with an undirected weighted graph having C vertices to which an index between 0 and $C - 1$ has been randomly assigned. C is the number of vertices in the first pass, and the number of communities found in previous pass otherwise.
- **Second phase: aggregation.** The second phase consists in building a new weighted graph whose vertices are the C communities found during the first phase.

The Louvain algorithm



Each pass of the Louvain method is made of two phases. The first phase (Vertex Mover) consists in sequentially moving vertices to neighboring communities leading to a maximum increase of modularity. The second phase aggregates vertices and constructs a meta-graph whose vertices are the communities found after the first phase. The two phases are repeated until no improvement of modularity is observed. In this illustration, the algorithm converges after two passes, uncovering an optimal partition made of 2 communities, and a value of $Q = 0.45$. In this example, it is easy to show that aggregating further the two communities into one big community would result in a decline of modularity, as $Q = 0$ in that case.

The Louvain algorithm

Algorithm 1 SingleVertexMover

Require: $G = (V, E, w)$ a weighted graph
Require: P a partition of V ($P[i]$ = community of vertex i)
Ensure : ΔQ, the gain of modularity
begin
  $c_{\text{old}} \leftarrow P[i]$
  REMOVE(i, c_{old})
  $C \leftarrow \{P[j] | (i, j) \in E\} \cup \{c_{\text{old}}\}$
  $c_{\text{new}} \leftarrow \arg\max_{c \in C}(\text{GAIN}(i, c))$
  if $\texttt{GAIN}(i, c_{\texttt{new}}) - \texttt{LOSS}(i, c_{\texttt{old}}) > 0$ **then**
    INSERT(i, c_{new})
    return $\texttt{GAIN}(i, c_{\texttt{new}}) - \texttt{LOSS}(i, c_{\texttt{old}})$
  else
    INSERT(i, c_{old})
    return 0

The Louvain algorithm

Because of its effectiveness and simplicity, the Louvain method has been extensively studied, and numerous modifications have been proposed, whether to improve results, further increase calculation speed or generalize its scope of use. We present these modifications in four different directions, each detailed in its respective subsection:

- the modification of the algorithm's core, including its initialization, stopping criterion, vertex mover, etc;
- the use of quality functions other than modularity, functions which may be similar to modularity but with other parameters, or functions which may be completely different;

Algorithm 2 VertexMoverIteration

```
Require: G = (V, E, ω) a weighted graph
Require: ϵ a stopping criterion
Ensure : a partition P of V
begin
    INIT(P) do
        increase ← 0
        forall vertices i do
            increase ← increase + SingleVertexMover(i)
    while increase > ϵ
    return P
```

The Louvain algorithm

Algorithm 3 Louvain algorithm

Require: $G = (V, E, \omega)$ a weighted graph
Ensure : a partition P of V
begin
  repeat
    $P \leftarrow$ VertexMoverIteration(G)
    $G \leftarrow$ PartitionToGraph(P, G)
  until no improvement is possible

- generalizations of Louvain to deal with more than simple graphs. These may include weighted, oriented, spatially constrained, temporal network, etc;
- and finally the use of parallelism to speed up computation time.

The Girvan-Newman algorithm

1. Algorithm description

The Girvan-Newman algorithm works based on the idea that edges with high "betweenness" centrality are likely to be "bridges" between communities, and removing them will help to reveal the community structure.

Specifically, the algorithm has **two central features** that distinguish it from those that have preceded:

- First, a "divisive" technique is used which iteratively removes edges from the network, thereby breaking it up in communities. The edges to be removed are identified by using one of a set of edge betweenness measures.
- Second, the algorithm includes "a recalculation step" in which betweenness scores are re-evaluated after the removal of every edge. This step turns out to be of primary importance to the success of its. Without it, the algorithms fail miserably at even the simplest clustering tasks.

The algorithm's steps for community detection are summarized below:

1. The betweenness of all existing edges in the network is calculated first.
2. The edge(s) with the highest betweenness are removed.
3. The betweenness of all edges affected by the removal is recalculated.
4. Steps 2 and 3 are repeated until no edges can be removed without disconnecting the network.

The Girvan-Newman algorithm

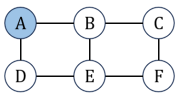
2. Betweenness measure calculation

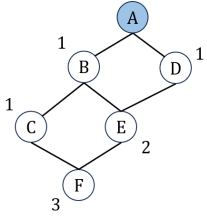
The betweenness measure is some measure that favors edges that lie between communities and disfavors those that lie inside communities. For most problems, a measure called **Shortest-path betweenness**, recommended by the authors to calculate the betweenness scores, which gives results about as good or better with considerably less effort:

1. From n source vertices of an undirected graph G, select one vertex to be a starting root node X and perform **breadth-first search** to find number of shortest path from the node X to each node, and assign the numbers as score to each node.
2. Starting from the leaf nodes, to the edge from vertex i to vertex j, with j being farther from the root node X than i, we calculate the weight of edge by 1 plus the sum of the weights on the neighboring edges immediately below it, all multiplied by $\frac{w_i}{w_j}$, with w_i and w_j are the assigned scores of i node and j node, respectively.
3. Compute the weights of all edges in the graph, and repeat from step 1 until all of the source vertices are selected.
4. Sum up all of the edge weights we compute in step 2 of n selected vertices and divide by 2 (to make it match with an undirected graph), and the result is the edge betweenness of edges.

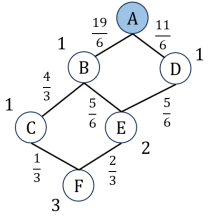
The Girvan-Newman algorithm

2. Betweenness measure calculation

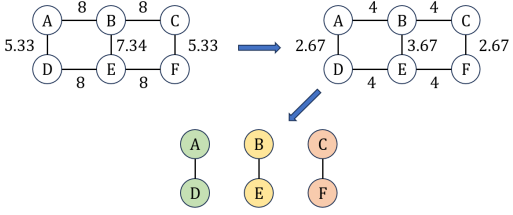




	A	B	C	D	E	F	Sum
AB	19/6	...	...	...	...	...	...
AD	11/6	...	...	...	...	...	...
BC	4/3	...	...	...	...	...	...
BE	5/6	...	...	...	...	...	...
CF	1/3	...	...	...	...	...	...
DE	5/6	...	...	...	...	...	...
FE	2/3	...	...	...	...	...	...



$$CF = 1 \times 1/3 = 1/3$$
$$FE = 1 \times 2/3 = 2/3$$
$$BC = (1 + 1/3) \times 1 = 4/3$$
$$BE = (1 + 2/3) \times 1/2 = 5/6$$
$$DE = (1 + 2/3) \times 1/2 = 5/6$$
$$AB = (1 + 4/3 + 5/6) \times 1 = 19/6$$
$$AD = (1 + 5/6) \times 1 = 11/6$$



The Girvan-Newman algorithm

3. Quantifying the strength of community structure

In order to know when the communities found by the algorithm are good ones, the same measure of the quality of a particular division of a network called **modularity**, which is also used by Louvain algorithm. If the number of within-community edges is no better than random, we will get $Q = 0$. Values approaching $Q = 1$, which is the maximum, indicate strong community structure. In practice, values for such networks typically fall in the range from about 0.3 to 0.7. Higher values are rare.

4. Algorithm limitation

The primary remaining difficulty with the algorithm is the relatively high computational demands it makes. The fastest of all suggested measures in the paper, the one based on shortest-path betweenness, operates in worst-case time $O(m^2 n)$, or $O(n^3)$ time on a sparse graph, which makes it usable for networks up to about 10000 vertices, but for larger systems it becomes intractable.

Dataset

Files:

- **meta.dimacs10-polbooks** – Metadata about the network.
- **out.dimacs10-polbooks** – The adjacency matrix of the network in whitespace-separated values format, with one edge per line. The meaning of the columns in **out.dimacs10-polbooks** are:
  - **First column**: ID of from node
  - **Second column**: ID of to node

Data samples:

- 1, 2
- 1, 7
- 2, 4
- 2, 6
- 2, 7
- ...

As can be seen from the data samples, the line ”1, 2” means that book 1 and book 2 are frequently co-purchased together.

Process the dataset

Function **read-dimacs**: Read raw data from file and transfer to a list. Each element of the list is a list of 2 elements, each element is the value of each column

---

procedure read-dimacs(*path*) `// path of the dataset`
    Open and read the dataset with the *path* then set to *f*
    Read all lines from *f* then set to *lines*
    Initial the dataset array as *data*
    Loop each *line* from *lines*
        Remove any leading and trailing whitespaces from *lines* then split *line* into a list
        Append *line* to *data*
    return *data*
end procedure

Process the dataset

Function **convert-to-graph**: Convert data to graph. We have to convert the data from **read-dimacs** output to **Graph** type from **network** package to use in **python-louvain** package as input, which has 2 steps. First, we get a list of non-duplicate nodes then add nodes to graph. Second, we add edges to graph, each edge is the value of each column.

---

procedure convert-to-graph(*data*) `// data is the output of the READ-DIMACS function`
    Initial the **Graph** object from **networkx** package as *G*
    Get a list of non-duplicate nodes from *data* then set to *allMembers*
    Append *allMembers* to *G*
    For each data point from *data*
        Add edge to *G* from *data*
    return *G*
end procedure

Louvain algorithm

In this step, we just need to load the dataset from **read-data.py** module, then use the functions from **python-louvain** package to compute the best partition. Then, we draw the graph and color the nodes according to their partition.

---

procedure Main process
  Load dataset from **CONVERT-TO-GRAPH** function then set to *G*
  Compute the best partition from *G* then set to *partition*
  Draw the graph from *G* then set to *pos*
  Color the nodes according to their partition from *pos*
end procedure

Louvain algorithm

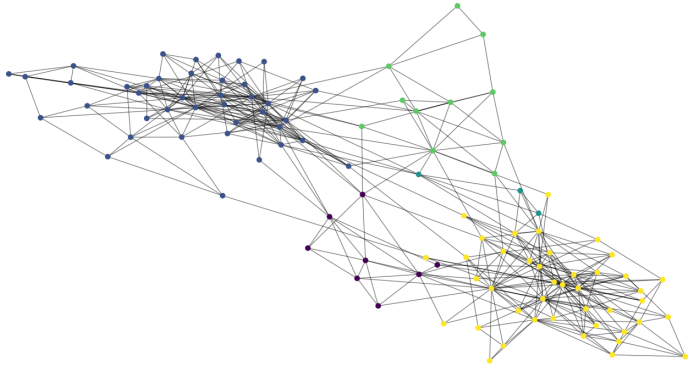


Figure: Result of 5 communities in 0.144s with modularity of 0.526789

Girvan-newman algorithm

In this step, we just need to load the dataset from **read-data.py** module, then use the **girvan_newman** community function from **networkx** package to compute the best partition. Then, we draw the graph and color the nodes according to their partition.

---

procedure Main process
  Load dataset from **CONVERT-TO-GRAPH** function then set to *G*
  Compute the best partition from *G* then set to *partition*
  Draw the graph from *G* then set to *pos*
  Color the nodes according to their partition from *pos*
end procedure

Girvan-newman algorithm

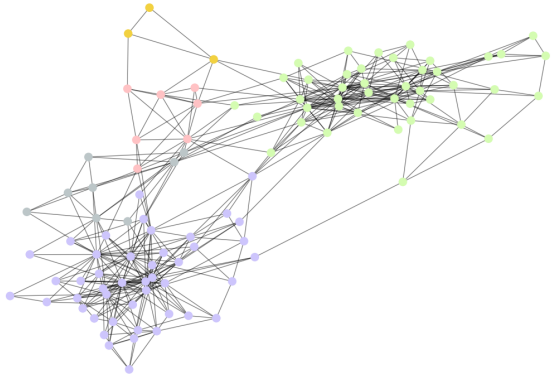


Figure: Result of 5 communities in 0.425s with modularity of 0.516801

Evaluation

The results of the Louvain algorithm are shown in the image. Vertices are colored according to the community they belong to.

Both Louvain and Girvan-Newman algorithms have a degree of randomness, so the results might be different on different runs.

- **Number of communities:** The number of communities detected by the algorithms after different runs is mostly around 4-5.
- **Modularity**: Most of the runs give value of modularity approximately around 0.53 and 0.52 for Louvain and Girvan-Newman respectively, which indicates a reasonably good community structure.
- **Community size**: Communities vary in size.
- **Connections between communities:** Communities are connected to each other by a small number of edges.
- **Running time**: Louvain algorithm runs faster than Girvan-Newman algorithm as expected, which is approximately 3 times faster.

Conclusion

In conclusion, both the Louvain and Newman-Girvan algorithms offer powerful solutions for community detection in complex networks. The Louvain algorithm excels in efficiently identifying community structures, revealing underlying connectivity patterns, owing to its iterative optimization of modularity. While, the Newman-Girvan algorithm offers a valuable approach to community detection by leveraging the concept of edge betweenness centrality.

However, it is important to acknowledge some limitations of the Louvain and Girvan-Newman algorithms. The Louvain algorithm's local optimization may lead to sub-optimal solutions, especially in networks with varied community sizes or hierarchical structures, while the Newman-Girvan algorithm's computational intensity in calculating betweenness centrality poses challenges for large-scale networks.

For the use case experiment **Customer network** with Political books dataset, based on different metrics, such as visualization, number of communities, community size, modularity, and connections between communities, we can see that both Louvain and Girvan-Newman algorithms succeed in detecting the communities and dividing the networks into different independent clusters with reasonable community numbers based on the visualization results.

Thanks for your attention