Derivations of Louvain Method Equations for Community Detection in Directed and Undirected Graphs

Joonyoung Bae[a,1]

[a] *University of Southern California, n99joon@gmail.com*

Abstract—This is to summarize all the equations related to the Louvain Method in a single document

Keywords—*Louvain Method, Directed Louvain Method*

Contents

1. Algorithm of Louvain Method

Initialization : Each node is in its own community
Algorithm:

1. Randomly sort the nodes (The order affects the resulting partition)
2. Loop through the nodes in the sorted order and **pop it from its original community** and **merge it to the neighboring community** that improves the modularity the most after merging. (Stay in original community if the change in modularity is negative)
3. If, after Step 1 and 2, modularity has increased (Or increased by more than the hyper-parameter **threshold**), merge all nodes in the same community into a single node and update all the edge weights accordingly. (The edges inside the same community become one self-loop edge of weight equal to the total sum of edge weights) (Inter-community edges between two communities merge into one with corresponding weight). Then repeat step 1 and 2. Otherwise, stop.

2. Undirected Louvain Method

2.1. Modularity (Q)

$$Q = \frac{1}{2m}\sum_i\sum_j (A_{ij} - \frac{d_i d_j}{2m})\delta(c_i, c_j)$$

- Q = Modularity of the graph G
- A_{ij} = Entry i x j or Adjacency Matrix (Can be more than 1 in graph with weighted edges)
- d_i, d_j = the degree of node i and j
- c_i, c_j = communities of node i and j
- $\delta(c_i, c_j)$ = Kronecker Delta Function

$$\delta(c_i, c_j) = \begin{cases} 1 & \text{if } c_i \text{ and } c_j \text{ are the same cluster} \\ 0 & \text{otherwise} \end{cases}$$

2.2. Modularity of Each Cluster in Sum of Degree Format (Q_c)

$$Q_c = \frac{1}{2m}\sum_i\sum_j (A_{ij} - \frac{d_i d_j}{2m})\mathbb{1}(c_i = c_j = c)$$

$$= \frac{1}{2m}\sum_i\sum_j A_{ij}\mathbb{1}(c_i = c_j = c) - \frac{1}{2m}\sum_i\sum_j \frac{d_i d_j}{2m}\mathbb{1}(c_i = c_j = c)$$

$$= \frac{1}{2m}\sum_i\sum_j A_{ij}\mathbb{1}(c_i = c_j = c) - \frac{1}{2m}\sum_i d_i\mathbb{1}(c_i = c)\frac{1}{2m}\sum_j d_j\mathbb{1}(c_j = c)$$

$$= \frac{\Sigma_{in}^c}{2m} - (\frac{\Sigma_{tot}^c}{2m})^2$$

- Σ_{in}^c = the sum of edge weights between nodes within the community c (each edge is considered twice)
- Σ_{tot}^c = the sum of all edge weights for nodes within the community c (including edges which link to other communities).

2.3. $Q = \sum_c Q_c$

$$Q = \sum_c Q_c$$

Because every node pair from different communities do not contribute to the Modularity due to $\delta(c_i, c_j)$, we can calculate separately and add them up.

2.4. Change of Q (ΔQ) of node i changing community from c_i to c_j

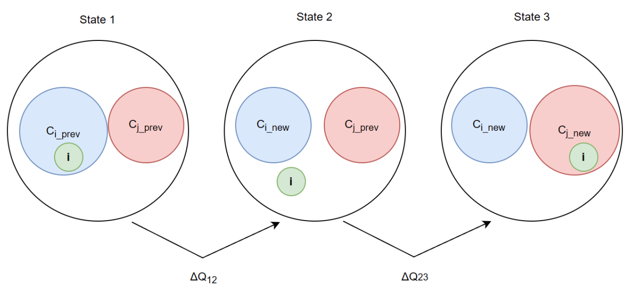

Figure 1. Node i leaving c_i and joining c_j

$$\Delta Q = \Delta Q_{12}\ \text{(Removal Cost)} + \Delta Q_{23}\text{(Merging Gain)}$$

2.4.1. ΔQ_{12} *(Removal Cost)*

$$\Delta Q_{12} = (Q_{c_{i\_new}} + Q_{node-i}) - (Q_{c_{i\_prev}})$$

$$= (\frac{\Sigma_{in}^{ci\_new}}{2m} - (\frac{\Sigma_{tot}^{ci\_new}}{2m})^2 + \frac{\Sigma_{in}^{node-i}}{2m} - (\frac{\Sigma_{tot}^{node-i}}{2m})^2) - (\frac{\Sigma_{in}^{ci\_prev}}{2m} - (\frac{\Sigma_{tot}^{ci\_prev}}{2m})^2)$$

$$= (\frac{\Sigma_{in}^{ci\_prev} - d_i^{ci\_prev}}{2m} - (\frac{\Sigma_{tot}^{ci\_prev} - d_i}{2m})^2 + 0 - (\frac{d_i}{2m})^2) - (\frac{\Sigma_{in}^{ci\_prev}}{2m} - (\frac{\Sigma_{tot}^{ci\_prev}}{2m})^2)$$

$$= -\frac{d_i^{ci\_prev}}{2m} + \frac{d_i(\Sigma_{tot}^{ci\_prev} - d_i)}{2m^2}$$

Where

- d_i^c = The degree of node i of edges only within community c (Each undirected edge is double counted for degree)
- d_i = Total degree of node i

The code inside the networkx python library (networkx.algorithms.community.louvain):

```
def _one_level(G, m, partition, resolution=1, is_directed=False, seed=None):
    ...
    degree = degrees[u] #d_i
    Stot[best_com] -= degree # (Sum_total - d_i)
    remove_cost = -weights2com[best_com] / m + resolution * (
        Stot[best_com] * degree
    ) / (2 * m**2)  # weights2com[best_com] = d_i^{c}/2
    ...
```

Code 1. Snippet from networkx library.

2.4.2. ΔQ_{23} *(Merging Gain)*

$$\Delta Q_{23} = (Q_{c_j\_new}) - (Q_{c_j\_prev} + Q_{node-i})$$

With similar calculation,

$$= \frac{d_i^{cj\_new}}{2m} - \frac{d_i(\Sigma_{tot}^{cj\_new} - d_i)}{2m^2}$$

The code inside the networkx python library (networkx.algorithms.community.louvain):

```
def _one_level(G, m, partition, resolution=1, is_directed=False, seed=None):
    ...
    gain = ( #gain = Delta Q
        remove_cost # Q_12
        + wt / m # wt = d_i^cj/2
        - resolution * (Stot[nbr_com] * degree) / (2 * m**2)
        )
    ...
```

Code 2. Snippet from networkx library.

3. Directed Louvain Method

4. References

Networkx louvain library
Networkx modularity library
Paper of Louvain Method
Paper of Directed Louvain Method
Louvain Method Wikipedia