Clustering Large Attributed Graphs: An Efficient Incremental Approach(Citation)

A brief presentation explaining the work of - - -

George Fakidis

Athens University of Economics and Business

June 2025

Conceptual Overview

- *Attributed Graphs*: We add a set of m attributes to the classical graph formulation(G=(V,E)). Each vertex is associated with an attribute vector.
- *Clustering*: we want to group graph nodes into separate clusters, useful for a variety of tasks. Like the popular k-means algorithm but the distance measure is specific to attributed graphs.
- *Objectives in attributed Graph Clustering*: Short distances within cluster, large distances between clusters. Similar attribute values of vertices within cluster, different attribute values of vertices between clusters.
- *Incremental*: Intermediate calculation results are used in the next iteration as a shortcut to avoid expensive recalculations.

Graph Model I

- *Attributed Graph Definition*: $G = (V, E, A)$, V is the set of vertices, E the set of edges, and A is the set of m attributes that are associated with vertices in V. Each vertex $v \in V$, is associated with an attribute vector $[a_1, \dots, a_m]$.
- *Attribute Augmented Graph Definition*: Basically introducing attribute vertices for each possible value of each attribute. Vertices associate themselves with attribute vertices through attribute edges.
- *Random Walk Distance*: The distance measure is a unification of structural distance(vertex edges) and attribute distance(attribute edges).

Graph Model II

- *Transition probability from node vertex to node vertex through structure edge*: (Pv)

$$\frac{W_e}{|N(v_i)| \cdot W_e + \sum_{i=1}^{m} W_{a_{im}}}$$

- *Transition probability from node vertex to attribute vertex through an attribute edge*: (A)

$$\frac{W_e}{|N(v_i)| \cdot W_{ea} + \sum_{i=1}^{m} W_{a_{im}}}$$

- *Transition probability from attribute vertex to node vertex(through attribute edge)*: (B)

$$\frac{1}{|N(v_i)|}$$

- If the respective edge,either structure edge or attribute edge,does not exist, the probability is simply 0.

Graph Model III

The transition probability matrix(calculated from the equations above) is:

$$P_A = \begin{bmatrix} Pv & A \\ B & O \end{bmatrix}$$

The O in the matrix are zeros as we cannot move from attribute vertex to another attribute vertex.

The Random Walk distance matrix is:

$$R_A = \sum_{i=1}^{l} c \cdot (1-c)^l \cdot P_A^l$$

SA-Cluster Algorithm Overview

A simple high level description of the algorithm:

1. Each vertex gets assigned to its closest centroid($O(n)$).
2. Update cluster centroids($O(n)$).
3. Adjust attribute edge weights $w1, \dots, w_m$.
4. Re-calculate the random walk distance matrix R_A(for walk

distance L: $O(L * n^3)$ due to matrix operations).

Repeat until convergence

The incremental approach and why it matters

- The expensive part of the algorithm is the random walk distance matrix calculation.
- Goal: Update R_A iteratively, using the $R_A prev$ and the weight increments $\{\Delta w_1, \ldots, \Delta w_m\}$. To do that we need to find $\Delta R_A prev$ and finally update the matrix using $R_A = R_A prev + \Delta R_A prev$.
- In $R_A prev$ we calculate only the changed elements, those affected by edge weight changes.

How the proposed algorithm works

The rest of the algorithm remains the same, here we see how we calculate the random walk matrix from one iteration to the next in an iterative manner to reduce the $O(L * n^3)$ cost of the original calculation.

- Calculate the base value for the iteration ΔP_A^1
- Calculate the increment and add it to the result

$$\Delta P_A^l = \begin{bmatrix} \Delta P_{V_l} & \Delta A_l \\ \Delta B_l & \Delta C_l \end{bmatrix}, \Delta R_A + = c(1-c)^l \Delta P_A^l$$

- Repeat for each step of the walk.
- Finally return $Rnext = R_A + \Delta R_A$.

Technical Details

- $\Delta PV_l = \Delta PV_{l-1} \cdot PV_1 + \Delta A_{l-1} \cdot B_1$
- $\Delta B_l = \Delta B_{l-1} \cdot PV_1 + \Delta C_{l-1} \cdot B_1$
- $\Delta A_l = [\Delta\omega_1 \cdot A_{l,a_1}, \ldots, \Delta\omega_m \cdot A_{l,a_m}] + \Delta PV_{l-1} \cdot AN_1$
- $\Delta C_l = [\Delta\omega_1 \cdot C_{l,a_1}, \ldots, \Delta\omega_m \cdot C_{l,a_m}] + \Delta B_{l-1} \cdot AN_1$
- $\Delta P_A^l = \begin{bmatrix} \Delta P_{V_l} & \Delta A_l \\ \Delta B_l & \Delta C_l \end{bmatrix}$

Theoretical Results

- The total number of zero elements in ΔP_A^2 is $O(n^2)$. This means that most elements do not change which entails that a small number of elements of the matrix need an update. Here, n_i is the size of the domain of attribute a_i. We assume that vertices are evenly distributed in the possible values of the attribute vector.
- Upper bound:

$$\frac{n^2 \times \prod_{i=1}^{m}(n_i - 1)}{\prod_{i=1}^{m} n_i}$$

- Lower bound:

$$\left(2n - \prod_{i=1}^{m} n_i\right) \times \prod_{i=1}^{m}(n_i - 1)$$

Experimental Results

- Runtime in Seconds is around 1/3 to 1/4 of the original non-incremental implementation.
- Cluster quality is the same, there are no significant differences.

Thank You!