Paper one

Fast Algorithm for Modularity-Based Graph Clustering

Motivation

Modularity-based graph clustering algorithm has become one of the most popular graph clustering method in recent years. However, since the existing algorithms need to scan all vertices or edges iteratively they are not capable to work on big data or large graphs. Therefore, the authors want to figure out a new creative and efficient method to solve this problem. The new method has to be efficient and work faster than the existing algorithms as well as be able to obtain a good result, which means that the output clusters are supposed to have high modularity.

Related Theories and Definitions

- Modularity: an evaluation of the clustering with the consideration of the density of edges inside clusters as compared to edges between clusters.

Its mathematical form is: $Q = \sum_u \left\{ \frac{e_{uv}}{2m} - \left(\frac{a_u}{2m} \right)^2 \right\}$ where the former term is going to represent the actual edges in a cluster u, and latter term is representing the expected edges in a cluster u.

- Modularity gain: $\Delta Q_{uv} = 2\left\{ \frac{e_{uv}}{2m} - \left(\frac{a_u}{2m} \right)\left(\frac{a_v}{2m} \right) \right\}$, which illustrates the modularity gain after combining two vertices u and v.
- The set of prunable vertices: the initialization of this set P is obviously zero, and if the vertex has only one neighbor or the cluster labels of its neighbors are the same, then this vertex can be pruned, which means we don't need to reconsider it in the following iterations any more. Then the definition of P is as followed:

$$P_i = \begin{cases} \varnothing, & i = 0 \\ \{u : |\Gamma(u) = 1|\} \cup \{u : \forall v, w \in \Gamma(u), c_v = c_w\}, & i > 0 \end{cases}$$

- The set of target vertices: the definition of the set of target vertices is easily understood since the vertices is either the pruned one or the target vertex:

$$T_i = \begin{cases} V, & i = 0 \\ T_{i-1} - P_i, & i \geq 1 \end{cases}$$

Main Algorithm

The authors give a new algorithm with three main ideas: Incremental aggregation & Incremental pruning & Efficient vertex selections:

Incremental Pruning:
According to pruning set definition, whether it has only one neighbor or its neighbors have same label, they have already been aggregated into only one vertex.

Algorithm 1 Graph clustering

Input: $\mathbb{G} = \{\mathbb{V}, \mathbb{E}\}$;
Output: clustering result of $\mathbb{G}$;
1: $i = 0, \mathbb{P}_0 = \emptyset, \mathbb{T}_0 = \mathbb{V}$;
2: **while** $|\mathbb{T}_i| > 0$ **do**
3: $i = i + 1$;
4: $\mathbb{P}_i = \{u : |\Gamma(u)| = 1\}$;
5: **for** $\forall u \in \mathbb{P}_i$ **do**
6: aggregate vertex u into its neighbor;
7: **end for**
8: $\mathbb{T}_i = \mathbb{T}_{i-1} - \mathbb{P}_i$;
9: select vertex u from $\mathbb{T}_i$ that has the smallest degree;
10: $v = \arg\max_{v'} \triangle Q_{uv'}$;
11: **if** $\triangle Q_{uv} > 0$ **then**
12: aggregate u and v into a single vertex w;
13: $\mathbb{T}_i = \mathbb{T}_i - \{u, v\}$;
14: $\mathbb{T}_i = \mathbb{T}_i \cup \{w\}$;
15: **else**
16: $\mathbb{T}_i = \mathbb{T}_i - \{u\}$;
17: **end if**
18: **end while**

Efficient Selection:
Under the power-law distribution, the vertex of smallest degree has the biggest probability of having the high modularity.

Incremental Aggregation:
Find the candidate vertex whose modularity gain is the largest. If the gain is positive, aggregate them into a new vertex otherwise just prune it.

When there is no vertex in Ti, the algorithm terminates, whose output is aggregated vertices. All vertices included in an aggregated vertex have the same cluster label.

My Summary

The algorithm handles the problem of working on large scale data which existing algorithms based on modularity cannot solve. The most important idea of the algorithm is to use increment way to deal with big data, thus it needs to redefine the content of modularity which is supposed to be on the vertices not on the clusters. Moreover, the algorithm uses pruning to work more efficiently, and those vertices pruned will be no longer considered in later iterations. The algorithm also uses a smart way to choose the aggregation candidate following the power-law distribution during each iteration. And the experiment results show that the algorithm introduced by this paper not only can work efficiently on big dataset but also has a good result where the clusters have high modularity.

Paper two

On the Permanence of Vertices in Network Communities

Motivation

In the area of community detection, the measurement of community remains two problems: whether a community is actually modular or not and how resilient the community structure is if some disturbances are introduced in it. In the past, the optimization methods either only consider the degrees inside the cluster or consider all the external links equally. Therefore, the authors want to come up with a good estimate to quantitatively measure about the community-like cluster of a network, which is not based the total number of outside links but based on the maximum external connections only.

Main Theory

This paper introduces a novel vertex-based scoring function called permanence, which will have fewer tie-breaking thus more accurate in its optimization process. It considers the following factors:

- External connections

Authors think that the consideration of the sum of all external links in old method is treating all the external connections equally. But it is the fact that only the "strongest pull" from a cluster is decisive. Therefore, the new measure which is called permanence is going to consider the maximum number of external connections for any single community.

- Internal connections

To represent the tightness of vertices more accurately, the authors come up with an dear which not only focus on the connections inside a cluster but also introduce a coefficient factor to act as the penalty term.

The mathematical formal is like that：

$$Perm(v) = \Big[\underbrace{\frac{I(v)}{E_{max}(v)}}_{\text{F1}} \times \underbrace{\frac{1}{D(v)}}_{\text{F2}}\Big] - \Big[\underbrace{1 - c_{in}(v)}_{\text{F3}}\Big]$$

Where I(v) is the number of internal connections with this vertex v, Emax(v) is the maximum number of external connections in a cluster with vertex v, D(v) is the degree of this vertex and cin(v) is the coefficient factor.

The permanence is between -1 and 1, where 1 means strong connectivity and -1 means weak connectivity when I(v)<<D(v). Specially, when permanence is equal to 0, the vertex has the equal "pull" from other clusters thus it can be considered as a unique cluster.

Optimization Algorithm

According to the experiment results in this paper, permanence is effective as a scoring function and is also sensitive to perturbations, which is exactly what we want. Therefore, the authors introduce a heuristic algorithm to maximize the permanence:

```
Input: A graph G.
Output: Permanence of G; Detected communities

  procedure Max Permanence(G(V, E))
      Each vertex is assigned to its seed community
      Set value of maximum iteration as maxIt
      vertices ← |V|
      Sum ← 0
      Old_Sum ← −1
      Itern ← 0
      while Sum ≠ Old_Sum and Itern < maxIt do
          Itern ← Itern + 1
          Old_Sum ← Sum
          Sum ← 0
          for all v ∈ V do
              (compute current permanence of v)
              cur_p ← Perm(v)
              if cur_p == 1 then
                  Sum ← Sum + cur_p
                  continue;

              cur_p_neig ← 0          ▷ Neig(v)=set of neighbors of v
              for all u ∈ Neig(v) do
                  (compute current permanence of u)
                  cur_p_neig ← cur_p_neig + Perm(u)

              ▷ Comm(v) is the set of neighboring communities of v
              for all C ∈ Comm(v) do
                  Move v to community C
                  (compute permanence of v in community C)
                  n_p ← Perm(v)

                      ▷ Neighbors of v are affected for this movement
                  n_p_neig ← 0
                  for all u ∈ Neig(v) do
                  (compute new permanence of u)
                      n_p_neig ← n_p_neig + Perm(u)
                  if (cur_p < n_p) and (cur_p_neig < n_p_neig) then
                      cur_p ← n_p
                  else
                      replace v to its original community
              Sum ← Sum + cur_p
      Netw_perm = Sum/vertices                ▷ Permanence of G
      return Netw_perm
```

My Summary

This paper gives us a new evaluation named permanence to determine whether the cluster is modular or not. From the experiment in this paper, we can see that permanence is more effective and sensitive to disturbance. And optimization algorithm is also given, whose basic idea is as following:

(1)Initialize every vertex as a single cluster;

(2)Move the vertex into some cluster if the permanence is larger, else remain original state.

(3)Repeat the process until the convergence is reached.