Task 1

1.1 Louvain Algorithm

The Louvain algorithm is an iterative algorithm to find community in graph by optimizing the modularity Q value:

$$Q = \frac{1}{2\mathrm{m}} \sum_{i \neq j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j)$$

Where A_{ij} is an element in the adjacent matrix in the network (if node i is connected with node j, then A = 1 else 0). C_i represents the community that node i located in. m is the total number of edges in the network. $\delta(C_i, C_j)$ = 1 if node i and j locate in the same community, else $\delta(C_i, C_j) = 0$. k_i represents the degree of node. $k_i = \sum_w A_{iw}$. Modularity is a measurement method to evaluate the division of a community network. Its physical meaning is the difference between the sum of the weights of the connected edges of nodes in the community and the sum of the weights of the connected edges under random conditions.

The Louvain algorithm mainly has two stages: First, Each point sets as a community. Then consider the neighbor nodes of each community and merge into the community. Find the largest positive modularity Q and merge the point to community. It could be better to repeat this stage many times until the modularity Q without changing. Second, the second phase of the algorithm consists in building a new network whose nodes are now the communities found during the first phase. To do so, the weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. Links between nodes of the same community lead to self-loops for this community in the new network.

Reference I: https://arxiv.org/pdf/2101.00922.pdf (my previous project report)

Reference II: https://www.jianshu.com/p/36525bee7aac

1.2 NMI Score

The NMI Score of Louvain algorithm on the given dataset is 0.5978

Task 2

2.1 Time complexity

In the first round of Louvain, it scans all nodes and append those nodes which could bring largest improvement of the modularity Q into communities. Then, the second step it folds the community found in the first step and take each community as a super node in the next round. The total complexity is O(|V|*log|V|).

2.2 Improvement

The scanning of Louvain takes most of time. And it would be great if a data structure is used to store the data. It can reduce the cost of each scan. Paper from https://downloads.hindawi.com/journals/mpe/2021/1485592.pdf proposed a dynamic iterative optimization method which can be used to reduce the cost of scanning. And a tree structure with proper splitting could reduce the time of forming community.

Task 3

A novel metropolis sampling algorithms for large scale graph data is proposed in (https://sci-hub.ru/https://ieeexplore.ieee.org/abstract/document/4781123/).

The problem of finding a 'representative subgraph sample' can be cast into the following optimization problem:

$$\text{argmin}_{S \sqsubseteq G \wedge |S|=n'} \ \Delta(\sigma(S), \sigma(G))$$

where G is the original graph, S is a subgraph of G with |S| = n' nodes, σ(X) is a topological property of graph X, and Δ is a distance function on these topological properties.

The Metropolis Algorithm can draw samples from any probability distribution, proved that a function $\varrho*$ proportional to the density $\varrho(S)$. However, in this paper, it chooses a $\varrho^*(S)$ that is inversely proportional to $\Delta_{G,\sigma}(S)^p$, where $\Delta_{G,\sigma}(S)$ can be any distance measure between topological properties of the sample S the original graph G, which is proved in the paper. The pseudocode is given:

Algorithm 1 Metropolis Subgraph Sampling

Input: Graph $G = (V, E)$, distance function $\Delta_{G,\sigma}(\cdot)$, sample size n', number of possible transitions $\#it$, exponent p

$S_{current} \leftarrow$ uniformly at random from G
$S_{best} := S_{current}$
for $i := 1$ to $\#it$ **do**
  node $v \leftarrow$ randomly from $S_{current}$
  node $w \leftarrow$ randomly from $(V \setminus S_{current}) \cup \{v\}$
  $S_{new} := (S_{current} \setminus \{v\}) \cup \{w\}$
  $\alpha \leftarrow$ uniformly at random from interval $[0, 1]$
  if $\alpha < (\frac{\Delta_{G,\sigma}(S_{current})}{\Delta_{G,\sigma}(S_{new})})^p$ **then**
    $S_{current} := S_{new}$
    if $\Delta_{G,\sigma}(S_{current}) < \Delta_{G,\sigma}(S_{best})$ **then**
      $S_{best} := S_{current}$
    end if
  end if
end for

Output: S_{best}