Digraph Clustering by Marginal Propagation

Haoran Zhang
Department of Computer Science, Duke University
hz271@duke.edu

Abstract

Community within graphs features both stronger similarities and differences within and between other communities. This paper approaches this problem differently by viewing community detection as a semi-supervised learning task, where the community labels are generated from marginal propagation. We test the approach on one graph dataset with ground-truth community labels. By comparing our method with true labels, the results show that though performance may not be optimal compared with agglomerative method, it predicts more abundant features that can be further utilized for overlapping community detection, and membership community similarity.

1 Introduction

Problem description: Graph clustering, also known as community detection, refers to the unsupervised detection of clusters/communities within a graph. A digraph (directed graph) is a graph that is made up of a set of vertices connected by directed edges [1]. The directed edges represent binary connectivity between nodes, which is especially useful when the relationships between them are asymmetric. Community detection predicts the membership information from the node-to-node relationship.

Significance: Communities reveal the hierarchical organization of nodes, such organization information are of great importance to studying a complex system like proteins structure, social networks, citation, etc [2].

2 Prior state of the art

Community detection exploration originated in the 1920s in the sociology field [3]. In 2002 [4], Girvan and Newman lead the community in a new direction with graph partition. Deep Learning based clustering methods is also being extensively developed in recent decades, where the representation of nodes is learned to predict communities [5]. Leiden algorithm solves the inherent problems on the early proposed Louvain algorithm of badly connected communities [6]. BlueRed method, initially proposed for undirected graph clustering, is also extended to the digraph problem [7].

3 Objective and method

Objective: A novel approach of viewing community detection as a semi-supervised learning task, where the community labels are generated by marginal propagation.

Method: We divide the clustering into five steps, with the details of the algorithm included in the **Appendix A**.

a. Remove the self-loops, isolated nodes from the graph (**Algorithm 1**);

Submitted to Prof. Xiaobai Sun in Compsci 521 Graph-Matrix Analysis.
GitHub Repo: https://github.com/Hollen0318/digraph_clustering_by_marginal_propagation

Table 1: Statistics for the test dataset Email EU Core

Dataset Statistics			
Nodes	1005	Edges in largest SCC	24729 (0.967)
Edges	25571	Average clustering coefficient	0.3994
Number of communities	42	Number of triangles	105461
Nodes in largest WCC	986 (0.981)	Fraction of closed triangles	0.1085
Edges in largest WCC	25552 (0.999)	Diameter (longest shortest path)	7
Nodes in largest SCC	803 (0.799)	90-percentile effective diameter	2.9

Table 2: Community Label distribution between the predicted and ground truth for Email EU Core

Dataset	**label**	**#**	**label**	**#**	**Dataset**	**label**	**#**	**label**	**#**	**label**	**#**	**label**	**#**
Email EU Core	**1**	263	**13**	8	**Ground Truth**	**1**	111	**13**	27	**25**	12	**37**	3
	2	135	**14**	8		**2**	107	**14**	26	**26**	10	**38**	3
	3	122	**15**	7		**3**	91	**15**	25	**27**	10	**39**	2
	4	82	**16**	5		**4**	56	**16**	24	**28**	9	**40**	1
	5	80	**17**	4		**5**	54	**17**	22	**29**	9	**41**	1
	6	42	**18**	4		**6**	49	**18**	19	**30**	8	**42**	0
	7	40	**19**	4		**7**	39	**19**	18	**31**	8		
	8	39	**20**	3		**8**	34	**20**	15	**32**	6		
	9	38	**21**	2		**9**	31	**21**	13	**33**	6		
	10	35	**22**	2		**10**	29	**22**	13	**34**	5		
	11	35	**23**	1		**11**	29	**23**	13	**35**	4		
	12	27				**12**	28	**24**	12	**36**	4		

b. Sum up the step-N (less or equal to the diameter) adjacency matrix (**Algorithm 2**);

c. Obtain zero values in both degree and whole matrix (**Algorithm 3**);

d. Filter unique mutually disconnected nodes (**Algorithm 4, 5**);

e. Label the marginal nodes as different communities and propagate (**Algorithm 6**);

f. Calculate modularity scores on predicted communities [8, 9];

4 Results

4.1 Clustering

Due to the computation limit, this work only tests the performance of the algorithm on the dataset Email-EU-Core, the statistics of which are listed in **Table 1**. By running the **Algorithm 6** for 100 epochs, we achieved the community label distribution summarized in **Table 2**.

4.2 Modularity

Various metrics are developed to measure the community detection performances, like DSC [10]:

$$(i)\ DSC(X,Y) = \frac{2 \cdot pu(X,Y) \cdot pu(X,Y)}{pu(X,Y) + pu(X,Y)}\ with\ pu(X,Y) = \frac{1}{n}\sum_{i} max_j |X_i \cap Y_j|. \quad (1)$$

For simplicity, we will use the conventional definition of modularity defined by Girvan *et al.* [4, 11]:

$$(ii)\ Modularity: \quad Q = \frac{1}{2m}\sum_{ij}(A_{ij} - P_{ij})\,\delta\,(C_i, C_j) \quad (2)$$

Q being the modularity score and m the total number of edges, P_{ij} the expected number of edges between vertices i and j in the null model. The δ-function returns one if vertices i and j are in the same community ($C_i = C_j$), zero otherwise.

The null model is defined as a replicate of the original graph while remaining some of its structural properties (like degree distributions) but without community structure. The choice of the null model is arbitrary with several possibilities existing. The configuration model proposes that the expected degree sequence of the null model should match the original graph. We choose a vanilla null model that only resembles the same degree distribution while the edges are rewired randomly.

The modularity scores Q for predicted clustering and ground truth are 0.275 and 0.316 respectively (with 100 EPOCHS, steps = 4 and community numbers = 23).

4.3 Membership Similarity

Because Marginal Propagation outputs a $N \times C$ matrix with N equals to the total number of nodes and C equals to the number of communities, these vectors describe the probability distribution of each node belonging to all communities. These membership vectors can be used to calculate membership similarity as in **Equation 3**:

$$Membership\ Similarity\ M_S = V_1 \cdot V_2 \tag{3}$$

4.4 Community Similarity

Given the membership probability distribution for each node, the similarity between communities can also be calculated. The dissimilarity scores between community i and j can be as simple as a fraction between two normalized summations of max probabilities p_{max} that contributes to the final clustering as in **Equation 4**.

$$Community\ Similarity = \frac{\sum_i^{C_i} p_j \cdot \sum_j^{C_j} p_i}{N(C_j) \cdot N(C_i)} \tag{4}$$

5 Conclusion

5.1 Abundant Features

Marginal Propagation gives the probability distribution of each node belonging to all communities, therefore it's capable of extracting more abundant features than simply community labels. Those probability distributions can be further utilized to calculate the similarity between nodes and communities. In addition, the distribution of membership probabilities can be applied in detecting overlapping-communities, e,g a linear combination of two community probabilities columns.

5.2 Limitations

The number of communities outputted is restricted by the diameter of the graph. That is to say, if the diameter of the graph is either too large or too short, this method's performance will be severely depreciated. Also, if the number of walks chosen is N, then the resulting communities will all have approximately half of that value, i.e. all communities will have about $N/2$ diameter.

If the diameter of the graph is either too large or too short, this method's performance will be severely depreciated. In addition, the community labels can be dominated by one node after long iterations, so the number of epochs (iterations) need manual design and experimentation.

We use the modularity score in **Equation 2** as an approximate measurement of community detection performance, however, the choose of Modularity Scores may not be optimal for our graph. Therefore, more accurate and appropriate metrics shall be considered in future studies.

5.3 Contributions

This project contributes to the community detection with an authentic proposed method named Marginal Propagation, due to limited time and resources, the optimal application of this method in terms of digraph clustering is not fully understood yet. For example, the impact of step size, number of iterations, probability initialization on clustering performance need to be further analyzed.

6 Acknowledgement

Appreciate Prof. Xiaobai Sun who has been supportive throughout the course.
Thanks to my friend & classmate Yuchen Jiang for her support and continuous encouragement.
Special gratitude to Zhiya Zhuo's github open source tool to calculate the modularity.

References

[1] Wikipedia contributors. Directed graph — Wikipedia, the free encyclopedia, 2022. [Online; accessed 12-December-2022].

[2] Arzum Karataş and Serap Şahin. Application areas of community detection: A review. In *2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT)*, pages 65–70, Dec 2018.

[3] Stuart A. Rice. The identification of blocs in small political bodies. *American Political Science Review*, 21(3):619–627, 1927.

[4] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[5] Xing Su, Shan Xue, Fanzhen Liu, Jia Wu, Jian Yang, Chuan Zhou, Wenbin Hu, Cecile Paris, Surya Nepal, Di Jin, Quan Z. Sheng, and Philip S. Yu. A comprehensive survey on community detection with deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2022.

[6] V. A. Traag, L. Waltman, and N. J. van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1):5233, 2019.

[7] Tiancheng Liu, Dimitris Floros, Nikos Pitsianis, and Xiaobai Sun. Digraph clustering by the bluered method. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2021.

[8] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[9] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Phys. Rev. Lett.*, 100:118703, Mar 2008.

[10] Dimitris Floros. Agreement or discrepancy measures. dec 2022.

[11] Santo Fortunato. Community detection in graphs. *Complex Networks and Systems Lagrange Laboratory, ISI Foundation, Viale S. Severo 65, 10133, Torino, I-ITALY.*, 486, feb 2010.

A Appendix

Algorithm 1: Remove the self-loops, isolated nodes in the graph (MATLAB)

Data: Graph Adjacency Matrix
Result: Self-loops, isolated nodes free adjacency matrix
(1) Load the data and obtain adjacency matrix as wells as graph object
A = Problem.A;
G = digraph(A);
(2) Remove the self-loops for all the nodes
diag = diag(diag(A));
A = A - diag;
(3) Remove the isolated nodes (indegree + outdegree = 0)
indeg = indegree(dig_A);
outdeg = outdegree(dig_A);
mixdeg = indeg + outdeg;
zero_pos = find(∼mixdeg);
G = rmnode(G, zero_pos);
A = adjacency(G);

Algorithm 2: Generate connectivity adjacency matrix after N steps/hops (MATLAB)

Data: Adjacency matrix without isolated nodes from **Algorithm 1**
Result: Pairwise connectivity matrix from one to N step/hops
(1) Calculate the connectivity matrix after N steps/hops
A_org = A;
A_sum = A_org;
while *N>0* **do**
    A = A*A_org;
    A_sum = A_sum + A;
    N = N - 1;
end
(2) Get the graph corresponding to the matrix
G = digraph(A_sum);

Algorithm 3: Find the positions of zeros in both types of degree and whole matrix (MATLAB)

Data: Pairwise connectivity matrix from one to N steps/hops from **Algorithm 2**
Result: Positions of zero values in degrees and whole matrix
A_Connectivity = (A_sum ∼= 0);
A_Connectivity_graph = digraph(A_Connectivity);
InDegree = outdegree(A_Connectivity_graph);
OutDegree = indegree(A_Connectivity_graph);
zeros = find(∼A_Connectivity);

Algorithm 4: Find unconnected nodes indexes after N steps/hops (MATLAB)

```
Data: Indexes of zeroes from Algorithm 3
Result: Mutual disconnected nodes indexes after N steps/hops
(1) Create all zeros matrix with same shape of A
Disc_matrix = zeros(length(A_1), length(A_1));
(2) Calculate the x and y coordinate of zero node
for i = 1:length(zero_pos) do
    x = ceil(zero_pos(i)/length(A_1));
    y = zero_pos(i) - (x-1)*length(A_1);
    if ismember(y, zero_pos_InDegree) | ismember(x, zero_pos_OutDegree) then
        continue;
    else
        Disc_matrix(x,y) = 1;
    end
end
(3) Keep only symmetric nodes in the matrix (i.e. when the disconnectivity is mutual)
total_zeros = sum(Disc_matrix, 'all');
Disc_matrix_sym = floor((Disc_matrix + Disc_matrix.')/2);
L = tril(Disc_matrix_sym);
L_total_num = sum(L, 'all');
Disc_node_id = find(L);
```

Algorithm 5: Record the positions of disconnected nodes and obtain unique list (MATLAB)

```
Data: Unconnected node indexes from Algorithm 4
Result: Indexes of unique nodes that are mutually unconnected
unique_disconnected_node = [0];
index = 1;
for i = 1:length(Dis_node_id) do
    x = ceil(Dis_node_id/length(A));
    y = Dis_node_id(i) - (x-1)*length(A);
    unique_disconnected_node(index) = x;
    unique_disconnected_node(index+1) = y;
    index = index + 2;
end
unique_nodes = unique(unique_disconnected_node);
The total number of communities assumed to be c:
c = length(unique_nodes);
```

Algorithm 6: Assign nodes into communities and update their probabilities (MATLAB)

```
Data: Unique disconnected nodes from Algorithm 5 and adjacency matrix A
Result: Probabilities of each node belong to Community from 1 to c
(1) Store the probabilities of each nodes belonging to community in a N × C matrix:
c_label = 1;
c_prob = zeros(length(A), length(unique_nodes));
for y = 1:length(A) do
    if ismember(y, unique_nodes) then
        c_prob(y, c_label) = 1.0
    else
        c_prob(y, :) = c_prob(y, :) + 1.0/length(unique_nodes);
    end
end
(2) Add community probabilities to neighbours proportional to its out-degree
EPOCHS = E;
for epochs = 1:EPOCHS do
    for node = length(A):-1:1 do
        sucs_nodes = successor(G, node);
        infl_prob = c_(node, :)
        for i = 1:length(sucs_nodes) do
            sucs_node = sucs_nodes(i);
            if ismember(sucs_node, unique_nodes) then
                continue;
            else
                c_prob(sucs_node, :) = c_prob(sucs_node, :) +
                 1.0*infl_prob/length(sucs_nodes)
            end
        end
    end
    for node = length(A):-1:1 do
        c_prob(node, :) = c_prob(node, :) / sum(c_prob(node, :));
    end
end
(3) We normalize all the probabilities to make their sums equal to one
[c_prob_max, c_label_pred] = max(c_prob, [], 2);
counts = hist(c_label_pred, [1 : C]);
```