Multiple Local Community Detection

I. Introduction

A common issue in the study of graph mining is community detection. We are interested in local community detection, whose goal is to locate communities that contain a particular collection of nodes, known as the seed set. The goal of our project is to detect multiple communities, given a set of nodes, which is called a seed set. To achieve this, an algorithm called MULTICOM is proposed by authors, which given seed set S can detect multiple communities nearby the nodes in seed set. On overview, the algorithm does the following things. It uses scoring metric personalized PageRank to define embedding of the graph round seed set. Based on the scoring conductance sweep cut on the graph is done to form community. After embedding DB Clustering Algorithm is used to cluster nodes which has same structural similarity. Based on the clusters, new nodes in neighborhood of initial seed set are picked as new seed nodes and algorithm will be able to derive multiple communities. We have Implemented the MULTICOM algorithm in java and able to get results similar to the results published in the paper.

II. Related Work

Our Algorithm MULTICOM, when compared with other algorithms in the papers we considered have some similarities and differences. [1] uses distance as metric to identify community and solves the problem in O(n^2). It forms community only based on distance obtained from random walk but not on structural similarity, which will be significant in social graphs. MULTICOM uses similarities of seed nodes to increase the community size. The algorithm defined in the paper [3] uses local modularity and vertex at a time discovery process to detect a community structure. If the graph has k explored vertices, then the algorithm takes a time of polynomial in k. This works a bit different from our MULTICOM as this algorithm works only to expand the seed set while MULTICOM also detects new seeds while expanding the local communities. It results in multiple communities getting detected by MULTICOM. If there are multiple communities having common nodes, then the post processing step of MERGE improves to output a single larger community. [5] is based LEMON algorithm, which is also used as a scoring function in MULTICOM, builds upon LEMON. The algorithm defined in the paper uses local modularity and vertex at a time discovery process to detect a community structure. If the graph has k explored vertices, then the algorithm takes a time of polynomial in k. This works a bit different from our MULTICOM as this algorithm works only to expand the seed set while MULTICOM also detects new seeds while expanding the local communities. It results in multiple communities getting detected by MULTICOM. If there are multiple communities having common nodes, then the post processing step of MERGE improves to output a single larger community.

III. Implementation

Algorithm 1 MULTICOM:

Require: Graph *G = (V, E)*, seed set S subset of Vertices V. parameters alpha = 0.85, I(number of communities) = 10

```
1: S = all seed nodes (null initially)
2: $S_{new}$ = S (new seed nodes)
3: C = [] (list of all communities)
4: **while** $S_{new}$ != null and len(C) <= I do:
5:     **# Detecting Communities from seed nodes**
6:     **for** s in $S_{new}$ do:
7:         $f_s$ = Page Rank Values for each node (computed using **personalized page rank
           technique**)
8:         $C_s$ = Local Community (Cut made using **Conductance Sweep Cut**)
9:         C.add($C_s$)
10:    **End** of for loop
11:    **# Embedding and Clustering**
12:    $x(v) = f_s(v)$ (list of scores w.r.t v)
13:    $D1, D2, ...., D_k$ = clustering($X_u$) (clusters of nodes are formed using **DB Clustering**
       Algorithm)
14:    **# Picking new seed nodes**
15:    $S_{new}$ = null
16:    E = explored nodes
17:    **for** k = 1, …, K do:
18:        **if**(numbers of nodes in $D_k$ also in E > alpha * len($D_k$)) **then**
19:            $s_{new}$ = node with highest degree in $D_k$
20:            $S_{new}$.add($s_{new}$)
21:        **End** if
22:    **End** for loop
23: **End** while loop
24: return C (communities)
```

Loading Graph: Undirected graph is created by reading file which consists of edges. The node connections are stored in the form of adjacency list.

Seed Set: A random node is added to seed set to start the algorithm and new nodes are added to the set at each iteration.

Approximate Personalized Page Rank (PPR): Page rank (node score) values are calculated, which are later used for sweep cut of communities and deriving new seed nodes.

Algorithm 2 PPR:

Require: Adjacency list, seed set, alpha = 0.85, epsilon = 10^-5

```
1: P = vector of page rank values
2: R = vector of current scores
3: **for** seed in seed set:
4:     seedValue = 1.0/len(seed set)
5:     R[seed] = seedValue
6: **End** of for loop
7: nextNodes = seed set (nodes for which Scores has to be calculated)
```

```
# Calculating score for nodes
7: for node in nextNodes:
8:     nodeDegree = degree[node]
9:     pushValue = R[node] – 0.5 * epsilon * nodeDegree
10:    R[node] = 0.5 * epsilon * nodeDegree


11:    P[node] = P[node] + (1.0 - alpha) * pushValue
12:    putValue = alpha * push_val
13:    Neighbors = list of neighbors of node
       # Considering all neighbors of seed set for the next iteration
14:    for n in Neighbors:
15:        old_score = 0.0
16:        if(R.contains(n)):
17:            old-score = R[n]
18:        End if
19:        neighborScore = putValue/degree[n]
20:        threshold = epsilon * degree[n]
21:        if(neighborScore > threshold):
22:            nextNodes.add(n)
23:        End if
24:    End for loop
25:    End for loop
26:    return P
```

Conductance Sweep Cut: After calculating page rank score, we sort the nodes based on their scores. Sweep cut is made on the graph to form community.

Algorithm 3. Conductance Sweep Cut:

Require: node scores, adjacency list.

```
1: s_nodes = list of nodes sorted based on scores
2: best_sweep_cut = s_nodes[0] (add node with highest score in our community)
3: sweep_cut = list of nodes in community(null initially)
3: total_volume = sum of degrees of all nodes
4: volume = 0.0
# Calculating conductance score
5: for n in s_nodes:
6:     volume = volume + degree(n)
7:     neighbors = list of neighbors of node n
8:     if(neighbors present in sweep_cut already) then
9:         cut = cut – 1
10:    else:
11:        cut = cut + 1
12:    sweep_cut.add(n)
13:    conductance = cut/min(volume, total_volume - volume)
14:    if(conductance < best_conductance) then
15:        best_conductance = conductance
16:    best_sweep_cut = sweep_cut
17:    End of if
```

```
18: return best_sweep_cut (community)
19: **End** of for loop
```

IV. Evaluation and Results:

The quality of communities is calculated using Ground truth communities provided along with the dataset. The Ground truth communities are the real time communities. We compare the communities detected by MULTICOM algorithm with ground truth to calculate F1 score.

F1Score = **max**(count of nodes in detected communities which are also present in ground truth community)/size of(detected community)

Algorithm 4 F1Score:

Require: Ground truth communities (GC), communities detected by MULTICOM (C)

```
1: F1_scores = list of maximum f1 scores of each community
2: for c in C:
3:         for g in GC:
4:                 count = no of nodes in c also in GC
5:                 precision = count/len(c)
6:                 recall = count/len(GC)
7:                 f1 = 2 * precision * recall / (precision + recall)
8:                 f1_max = max(f1,f1_max)
9:                 F1_scores.add(f1_max)
10: End of for loop
11: return F1_scores
```

Datasets: We tested our algorithm on 3 real data sets. Amazon, DBLP and YouTube community datasets. The results are provided in a table comparing the results in original paper to our results.

DBLP DATASET:

https://snap.stanford.edu/data/com-DBLP.html

	F1 Score
Original Paper	0.23 ± 0.15
Our implementation	0.193 ± 0.14

Table 1: F1 score of original paper and our implementation on DBLP dataset

AMAZON DATASET:

Data available on: https://snap.stanford.edu/data/com-Amazon.html

	F1 Score
Original Paper	0.44 ± 0.19
Our implementation	0.45 ± 0.45

Table 2: F1 score of original paper and our implementation on Amazon dataset

YOUTUBE DATASET:

https://snap.stanford.edu/data/com-Youtube.html

	F1 Score
Original Paper	0.05 ± 0.03
Our implementation	0.05 ± 0.01

Table 3. F1 score of original paper and our implementation on YouTube dataset

V. Conclusion

Our implementation is successful in generating multiple local communities given a seed set. It can derive new seed nodes and form communities from the new seed nodes, thus performing better than tradition community detection algorithms. Compare to the results published in the main paper, our results are almost equivalent. The quality of communities is calculated by selecting 30 node from the graph at random from undirected graph.

VI. References

[1] P. Pons and M. Latapy. Computing communities in large networks using random walks. In International Symposium on Computer and Information Sciences,pages 284–293. Springer, 2005.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment, 2008(10):P10008, 2008.

[3] A. Clauset. Finding local community structure in networks. Physical review E, 72(2):026132, 2005.

[4] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using pagerank vectors. In Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on, pages 475–486. IEEE, 2006.

[5] Y. Li, K. He, D. Bindel, and J. E. Hopcroft. Uncovering the small community structure in large networks: A local spectral approach. In Proceedings of the 24th international conference on world wide web, pages 658–668. International World Wide Web Conferences Steering Committee, 2015