Comparison of Performance of Community Detection Algorithms Based on Modularity

Yuxuan DU
Electronic and Computer Engineering
University of Waterloo

Chengcheng JIANG
Management Science
University of Waterloo

ABSTRACT

Networks hold a significant importance in every domain of life. With the increase in networks we can observe more complex systems. Community structures are an essential part of such complex networks and therefore, the extraction of communities for studying the behavior and trends of individuals forming communities is of great use today. Here comes the community detection algorithms which play a vital role in detecting clusters. In this article, we implement two community detection algorithms in graphframes and compare the performance of different algorithms both in terms of running time and modulariy, which is a metric evaluating the structure of graph. Among these two algorithms, we also propose a simple sub-algorithm based on 'node betweenness', which performs poorly on the dataset.

INTRODUCTION AND METHODOLOGY

When it comes to a vertex's importance in a graph, centrality computation is one way to look at. Page rank, which is a focus of graph processing in this course, is one metric of measuring centrality. Inspired by this, we move forward to explore more about centrality metrics, including degree centrality, closeness centrality, betweenness centrality, etc. Among them betweenness centrality is to measure a node's effect on the information flow of a graph, which is often used to detect the 'bridge' between two sub-graphs, and one step further, to detect the different communities.

Followed by the above description, for this report, we would like to investigate into the performance differences of two different 'betweenness' algorithms to detect the community in a social network. Technically, we will first learn a new big data processing framework—the graphframes library (a.k.a graphX) in Spark, followed by implementing and analyzing two graph algorithms using this framework.

Our first algorithm is inspired by the "node betweenness" in the thesis 'A Faster Algorithm for Betweenness Centrality', which is the classic and time-efficient algorithm in the field of detection. In this algorithm, the overall idea is to use BFS (Breadth-first search) method to find the shortest path between all pairs of nodes in an unweighted graph, and use Dijkstra's algorithm to do so in a weighted graph. Then it examines for each pair (say s and t), how many shortest paths between s and t pass through node v. After summarizing all the ratios for each possible pair, the betweenness score of node v is calculated. The Pseudocode of this algorithm is below.

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

This algorithm requires $O(m+n)$ space and run $O(nm)$ and $O(nm+n^2 \log n)$ time on unweighted and weighted graphs respectively, where n is the number of nodes and m is the number of edges in a graph. [5]

The second algorithm is called 'Girvan–Newman algorithm'. Instead of calculating the betweenness score of a node, it extends this definition to the case of edges, defining the "edge betweenness" of an edge as the number of shortest paths between pairs of nodes that run along it. If there is more than one shortest path between a pair of nodes, each path is assigned equal weight such that the total weight of all of the paths is equal to unity.

After the betweenness score is calculated, we need to use it to detect potential communities. And the methods vary between these two different algorithms.

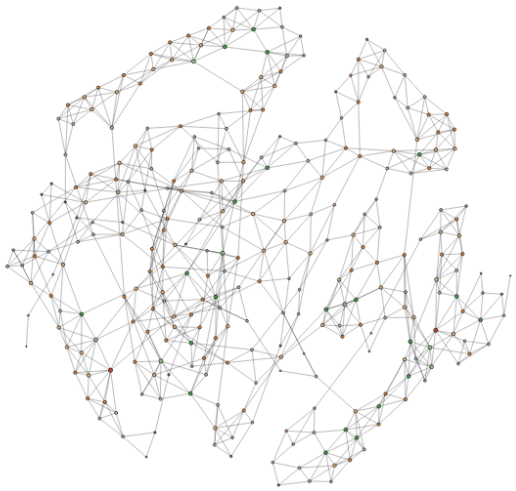
For the algorithm developed by Brandes, we adapt the betweenness calculation idea to community detection. We first rank the betweenness score in a descending order. Then we gradually delete the nodes with the highest betweenness score to form a new graph (in this process the relevant edges are deleted as well), and use the new graph to do connected component analysis. This process is repeated until the components of the graph become stable. For instance, we first delete nodes with the top 10 highest betweenness score, followed by deleting nodes with the top 20, 30, 40… highest score, and finally reach a state that the components of the original graph not changed. We would refer to this algorithm as "node-deletion algorithm".

For 'Girvan–Newman algorithm', community detection is a part of this algorithm. The basic logic is that the edges connecting communities will have high edge betweenness. By removing these edges, the groups are separated from one another and so the underlying community structure of the graph is revealed. More specifically, the steps are as follows: 1) the edges with the highest betweenness are removed; 2) the betweenness of all edges affected by the removal is recalculated; 3) the previous 2 steps are repeated until no edges remain.[3] However, for the sake of community detection we need to stop deleting edge before no edges left. And when to stop can be a tricky problem. For simplicity, we stop deleting edges when the number of communities left is equal to the number of communities indicated by our dataset's ground truth.

DATASET

To implement the two algorithms described above, we choose a dataset from an interactive scientific network data repository (http://networkrepository.com/). Since we would like to do community detection and evaluate the performance, we prefer graphs with ground-truth communities. That is to say, we are interested in picking a graph that has labels already, in which case we could compare our results with the original community labels. Also, the time complexity of the two algorithms implies that we should pick a small graph to avoid long running time.

Based on the above criteria, we choose a graph from the 'labels networks' on the repository. This graph is labelled as 'DD244' with 291 nodes and 822 edges. And each node has a label indicating which community it belongs to. Visualization of the graph:



We can see that this graph has a clustering structure which is helpful to community detection. With this graph, we will conduct analysis using different algorithms and evaluate the detection performance.

EVALUATION METRICS

Modularity: It is a measure on the structure of the graph. It basically describes how well the graph can be divided into clusters. The idea is that the higher proportion of edges falling in each cluster rather than inter-clusters, the more likely the graph can be divided into clusters accordingly.

$$Q = \sum_{i=1}^{k} \left(e_{ii} - a_i^2 \right)$$

where e_{ii} = probability of edges in module i and a_i = probability a random edge would fall into module i.
The detailed explanation is given in several papers.[1][2] We would use this metric to evaluate different algorithms.

Since the dataset we use has labels on each vertex, we would also evaluate algorithms based on these ground-truth. The label indicates there are 20 groups, which will be utilized in the stopping criteria for GN algorithm. (a side note: the label in the dataset may not be strongly correlated to the community structure, so it can only serve as a supplementary metric)

RESULT

We run the Node-deletion algorithm based on the previous description in the Methodology part. And GN algorithm is run until only 20 communities are left, which means we keep deleting edges until 20 clusters are left.

We use the modularity API provided by python-louvain library to compute the modularity score for each algorithm. The first one is a benchmark algorithm which maximizes the modularity of network, so we can compare the score of other algorithms with it. However, the problem of maximizing modularity of a graph is a NP-hard problem. It is not a feasible option for dealing with large graphs both in theory and in practice[6]. We also include label propagation which is a time-efficient but low-accuracy algorithm. Details of label propagation algorithm can be found here.[4]

Modularity of score of the result of each algorithm:

Benchmark(upper bound of performance): 0.8048377643987426

Girvan-Newman: 0.7756688037603376

Node-deletion: 0.3835172950669248

Label propagation: 0.682367053237904

As we can see, Girvan-Newman algorithm performs really well in terms of modularity score. However, the time consumed by GN algorithm is much longer than label propagation. (GN algorithm's running time is around one hour in practice while label propagation takes only few seconds. One reason might be that our graphframes implementation has not been optimized yet. For example,

the repeated recalculation of BFS significantly slows down the performance.) Node-deletion algorithm performs even worse than LPA in terms of time consumption and modularity score.

In regard to the node-deletion algorithm, the reason why it performs poorly might be that since we only compute BFS between each pair of vertices once, too much weight would be assigned to nodes which are already obvious "bridge nodes". Other nodes that are potentially bridges would be assigned too little weights because they are overshadowed by the "obvious bridges". As for the running time, the time consumed by the node-deletion algorithm is less than GN algorithm because it runs BFS between each pair of nodes only once. However, the improvement in time efficiency cannot compensate the degradation of accuracy. So we can safely draw the conclusion that the node deletion algorithm fails and the recomputing of betweenness scores after each removal of edges or nodes is necessary.

We also evaluate the cluster collections based on the ground truth, that is the label of each node. But it turns out that the label has weak links to the clusters' id. So it's a poor indicator of the cluster structure.

LIMITATIONS

One significant drawback of GN algorithm is that it is extremely time consuming. The time complexity of GN algorithm is $O(m^2n)$, where m is the number of edges and n is the number of vertices. For sparse graph time complexity degrades to $O(n^3)$. Meanwhile for label propagation the time complexity is $O(n)$. So it may be compulsory to apply label propagation to large graphs because we cannot afford the time consumed by GN algorithm.

Another limitation is that GN algorithm does not perform well on dense graph. That's because on dense graph it is unlikely that a few removals of edges would lead to separation of clusters. We also run GN algorithm on a denser graph[7] which has 1005 nodes and 25788 edges. and the resulting community collections involve many isolated vertices, which makes little sense in community detection.

FUTURE STUDY

In graphframes, using BFS to get the actual path from one node to the other is not time efficient. For our dataset, it takes several minutes to compute the shortest paths from a pair of nodes. Even with such a small graph, the time to compute all the shortest paths between all pairs is rather long. For further study, we may take the time efficiency into account and find more powerful framework that can handle such a high time complexity algorithm. We may also evaluate what is the optimal point of stopping removing edges in GN algorithm to maximize modularity.

REFERENCE

1.U. Brandes *et al*., "On Modularity Clustering," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 2, pp. 172-188, Feb. 2008.

2.Newman, Mark. (2006). Newman MEJ.. Modularity and community structure in networks. Proc Natl Acad Sci USA 103: 8577-8582. Proceedings of the National Academy of Sciences of the United States of America. 103. 8577-82. 10.1073/pnas.0601602103.

3.Girvan M. and Newman M. E. J., Community structure in social and biological networks, Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002) (also Wikipedia: https://en.wikipedia.org/wiki/Girvan%E2%80%93Newman_algorithm)

4.Raghavan, Nandini & Albert, Réka & Kumara, Soundar. (2007). Near Linear Time Algorithm to Detect Community Structures in Large-Scale Networks. Physical review. E, Statistical, nonlinear, and soft matter physics. 76. 036106. 10.1103/PhysRevE.76.036106.

5.Ulrik Brandes (2001) A faster algorithm for betweenness centrality, The Journal of Mathematical Sociology, 25:2, 163-177, DOI: 10.1080/0022250X.2001.9990249

6.U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, D. Wagner (2006). Maximizing Modularity is hard.

7.email-Eu-core network https://snap.stanford.edu/data/email-Eu-core.html