Community Detection in Networks

CS 3353 - Fall 2021 - Programming Assignment 03
Due: Friday Nov 12, 2021 @ 6am.

Description:

Community detection refers to the automated discovery of highly interconnected collections of nodes in a graph (or network). Consider, for example, a graph representing multiple groups of friends from Facebook. One would expect the number of interconnections (edges) between the members of each group to be higher than between nodes in other friend groups.

For this project, you'll implement the community detection algorithm proposed by Girvan and Newman in [1] and refined in [2]. The Girvan-Newman algorithm relies on repeatedly calculating **edge betweenness**, a value for each edge which represents the number of shortest paths between all pairs of nodes that travel through said edge. After calculating betweenness for all edges, the edge with the highest score is removed, and betweenness is recalculated for all remaining edges. It is a divisive hierarchical clustering algorithm and the results can be visualized with a dendrogram (see Figures 2, 4, and 5 of [1]).

Your implementation should make use of the adjacency list container and associated algorithms of the Boost Graph Library (BGL). However, you MAY NOT use the Brandes Betweenness Centrality(...) algorithm supplied by BGL.

What are you to do?

Completing the 3 items below will earn you no more than 65 points on this project. The remaining 35 points must be earned by extending this project in a direction of your own personal interests.

Checking the boxes (up to 65 points)

1. Implement the Girvan Newman Algorithm for undirected, unweighted graphs. This includes the ability to read in graphs from a file as well as write out the community structure to a separate file. You should focus on the GraphML structure since Boost already contains functionality to read in gml files.
2. Test your implementation with randomly generated graphs. In [1], see the section on *Computer Generated Graphs* on page 7823. Replicate this testing strategy for your implementation. You can generate graphs directly in C++ or you can use the Python NetworkX library to generate graphs. I encourage you to share your generated graphs with your colleagues in the class.
3. Use the Football Conference 2000 Dataset also used in [1] to further verify the validity of your implementation. You can find the dataset linked from http://www-personal.umich.edu/~mejn/netdata/

Extension Ideas (up to 35 points)

Here are some ideas for ways you could extend this project. You don't have to choose one of these, you are free to come up with your own extension to have some fun and wow me with your creativity.

1. Find the Google Scholar entries for [1] and/or [2]. At the bottom of entries (see Figure 1 for an example), you'll see the "Cited by …" link. When you click that link, you'll be taken to a collection of other scholarly work that has cited the original paper.
    a. Find a paper that describes a strategy for improving the efficiency of some aspect of the Girvan-Newman algorithm. Implement it. Compare and contrast the performance of your original implementation and the new-and-improved implementation. You may need to generate more / larger graphs to really stress test the new implementation.
    b. Find a paper that describes a competing algorithm for community detection. Implement it. Compare and contrast the communities generated as well as the efficiency of the algorithms (Girvan Newman and the new one). You can also find some alternative strategies here.
2. Can you scale your solution to really large graphs? Consider some of the data sets available at the Stanford Network Analysis Project. Consider modifying your implementation to use all the resources of your processor (all the cores) or moving your implementation to ManeFrame II and engage in some massive parallelization. How big of a graph can you eventually process?
3. <Insert your own creative idea for a way to extend this project here. Double check with Fontenot when you have an idea.>

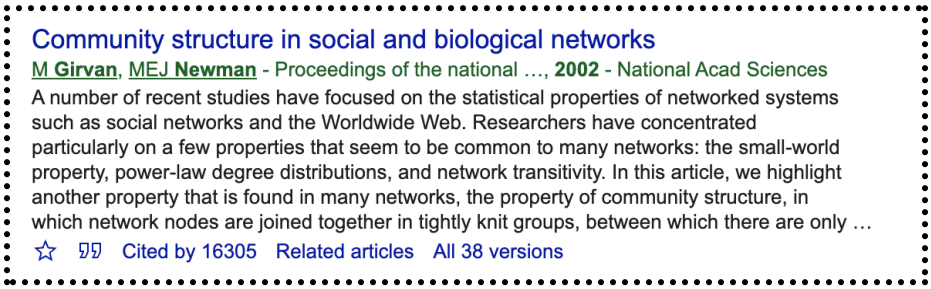

Figure 1 - Google Scholar entry for [1].

Your Deliverables

1. Your complete code base.
2. For the basic deliverables (1 - 3 on page 1), supply a project report in your team repo's README.md.
3. For your extension implementation, write a Medium.com blog post on what you did. More on this coming out soon!

Per usual, you can do this project individually or in teams of two.

Additional Resources:

- A nice explanation of the Girvan Newman Algorithm can be found here. It is part of NetworkX, a python library for working with large networks.
- Boost can be overwhelming at first due to its intense use of templating and generic programming. An excellent introduction to the BGL can be found in Boost.Graph Cookbook 1: Basics by Richel Bilderbeek. This reference starts literally with the most simple graph possible: an empty undirected graph, and iteratively builds up more complex graphs example by example. Pay particular attention to vertex and edge object bundling, which means adding an object that can hold data to both edges and/or vertices. There is also a Volume 2, but I don't think you'll need it.
- Mining of Massive Datasets - Section 10.2 is on point.
- Social and Information Network Analysis Course Slide Deck 14

References:

[1] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, Jun. 2002, doi: 10.1073/pnas.122653799.

[2] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," Phys. Rev. E, vol. 69, no. 2, p. 026113, Feb. 2004, doi: 10.1103/PhysRevE.69.026113.