Lab 2 – Community Finding

COMP-47270 Computational Network Analysis and Modelling

Author : *Paula Dwan*
paula.dwan@gmail.com

Student ID : *13208660*

Lecturer : *Dr. Neil Hurley*

Due Date : *Sunday, 19 April 2015*

Table of Contents

IGRAPH – PYTHON

Here is a short summary about the community detection algorithms currently implemented in igraph:

INTRODUCTION

7 community detection algorithms :

- **Edgebetweenness** (Girvan-Newman link centrality-based approach),
- **Walktrap** (Pons-Latapy random walk-based approach),
- **Leading Eigenvectors** (Newman's spectral approach),
- **Fast Greedy** (Clauset et. al modularity optimization),
- **Label Pr#opagation** (Raghavan et. al),
- **Louv#ain** (Blondel et. al, modularity optimization),
- Spinglass (Reichardt-Bornholdt, modularity optimization),
- InfoMap (Rosvall-Bergstrom, compression-based approach).
- Other related functions: process modularity, deal with hierarchical structures, etc.
- Available in R, C and Python

EDGE.BETWEENNESS.COMMUNITY

This a hierarchical decomposition process where edges are removed in the decreasing order of their edge betweenness scores (i.e. the number of shortest paths that pass through a given edge). This is motivated by the fact that edges connecting different groups are more likely to be contained in multiple shortest paths simply because in many cases they are the only option to go from one group to another. This method yields good results but is very slow because of the computational complexity of edge betweenness calculations and because the betweenness scores have to be re-calculated after every edge removal. Your graphs with ~700 vertices and ~3500 edges are around the upper size limit of graphs that are feasible to be analyzed with this approach. Another disadvantage is that `edge.betweenness.community` builds a full dendrogram and does not give you any guidance about where to cut the dendrogram to obtain the final groups, so you'll have to use some other measure to decide that (e.g., the modularity score of the partitions at each level of the dendrogram).

FASTGREEDY.COMMUNITY

This is another hierarchical approach, but it is bottom-up instead of top-down. It tries to optimize a quality function called modularity in a greedy manner. Initially, every vertex belongs to a separate community, and communities are merged iteratively such that each merge is locally optimal (i.e. yields the largest increase in the current value of modularity). The algorithm stops when it is not possible to increase the modularity any more, so it gives you a grouping as well as a dendrogram. The method is fast and it is the method that is usually tried as a first approximation because it has no parameters to tune. However, it is known to suffer from a resolution limit, i.e. communities below a given size threshold (depending on the number of nodes and edges if I remember correctly) will always be merged with neighboring communities.

WALKTRAP.COMMUNITY

This is an approach based on random walks. The general idea is that if you perform random walks on the graph, then the walks are more likely to stay within the same community because there are only a few edges that lead outside a given community. Walktrap runs short random walks of 3-4-5 steps (depending on one of its parameters) and uses the results of these random walks to merge separate communities in a bottom-up manner like `fastgreedy.community`. Again, you can use the modularity score to select where to cut the dendrogram. It is a bit slower than the fast greedy approach but also a bit more accurate (according to the original publication).

LEADING.EIGENVECTOR.COMMUNITY

This is a top-down hierarchical approach that optimizes the modularity function again. In each step, the graph is split into two parts in a way that the separation itself yields a significant increase in the modularity. The split is determined by evaluating the leading eigenvector of the so-called modularity matrix, and there is also a stopping condition which prevents tightly connected groups to be split further. Due to the eigenvector calculations involved, it might not work on degenerate graphs where the ARPACK eigenvector solver is unstable. On non-degenerate graphs, it is likely to yield a higher modularity score than the fast greedy method, although it is a bit slower.

LABEL.PROPAGATION.COMMUNITY

This is a simple approach in which every node is assigned one of *k* labels. The method then proceeds iteratively and re-assigns labels to nodes in a way that each node takes the most frequent label of its neighbors in a synchronous manner. The method stops when the label of each node is one of the most frequent labels in its neighborhood. It is very fast but yields different results based on the initial configuration (which is decided randomly), therefore one should run the method a large number of times (say, 1000 times for a graph) and then build a consensus labeling, which could be tedious.

IGRAPH INFO

iGraph 0.6 will also include the state-of-the-art Infomap community detection algorithm, which is based on information theoretic principles; it tries to build a grouping which provides the shortest description length for a random walk on the graph, where the description length is measured by the expected number of bits per vertex required to encode the path of a random walk.

Anyway, I would probably go with `fastgreedy.community` or `walktrap.community` as a first approximation and then evaluate other methods when it turns out that these two are not suitable for a particular problem for some reason.

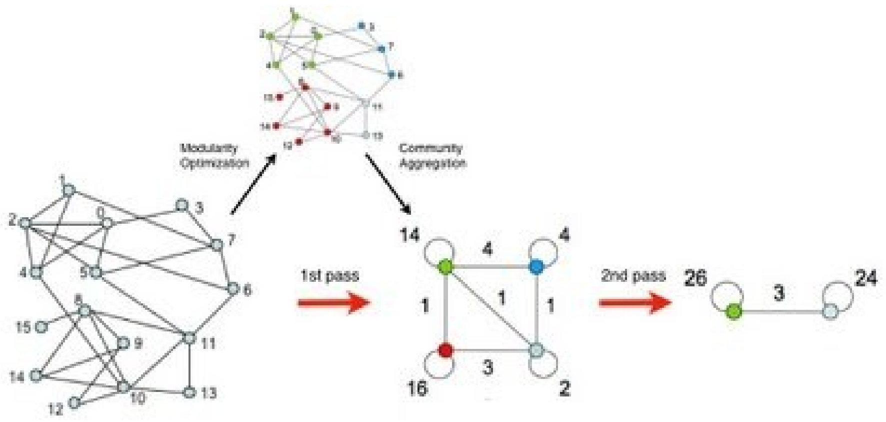
LOUVAIN METHOD: FINDING COMMUNITIES IN LARGE NETWORKS

A Multi-Level Aggregation Method for optimizing modularity

In the last few years, there have been many attempts to uncover communities in large networks. By large, we mean systems composed of millions of nodes, which cannot be visualized nor analyzed at the level of single nodes and therefore require a coarse-grained description.

Our method, that we call Louvain Method (because, even though the co-authors now hold positions in Paris, London and Louvain, the method was devised when they all were in Louvain), outperforms other methods in terms of computation time, which allows us to analyze networks of unprecedented size (e.g. the analysis of a typical network of 2 million nodes only takes 2 minutes). The Louvain method has also been to shown to be very accurate by focusing on ad-hoc networks with known community structure. Moreover, due to its hierarchical structure, which is reminiscent of renormalization methods, it allows to look at communities at different resolutions.

The method consists of two phases. First, it looks for "small" communities by optimizing modularity in a local way. Second, it aggregates nodes of the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained.



The output of the program therefore gives several partitions. The partition found after the first step typically consists of many communities of small sizes. At subsequent steps, larger and larger communities are found due to the aggregation mechanism. This process naturally leads to hierarchical decomposition of the network.#

This is obviously an approximate method and nothing ensures that the global maximum of modularity is attained, but several tests have confirmed that our algorithm has an excellent accuracy and often provides a decomposition in communities that has a modularity that is close to optimality.

The method was first published in: Fast unfolding of communities in large networks (http://arxiv.org/abs/0803.0476), Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, Journal of Statistical Mechanics: Theory and Experiment 2008 (10), P1000