TSKS11 Hands-On Session 3

Fall 2019

Task 1: Use SNAP and/or Gephi to Recognize Structural Properties of Networks

Study the files `g1.NET`, `g2.NET`, ... Analyze each one, and determine what network it is:

- a random (Poisson) network (if so, what parameters)?
- a scale-free network (if so, what parameters)?
- a regular graph?
- something else, if so what (if anything can be said)?

Questions to answer:

- Do the networks have any clearly identifiable community structure?
- What do their degree distributions look like? Plot both on linear, and log-log scale.
- What are their clustering coefficients?
- Do they show any small-world behavior? (Only qualitatively.)

The recommended approach is to first use Gephi to visualize the networks (note: one of them is very large) and have a first look at the degree distributions.

To export the degree distribution and local clustering coefficients to CSV (comma-separated values, can be read by Matlab for example), you can use the program `analysis.cpp`. To compile `analysis.cpp`, run `make analysis` on the command line. The compilation yields some warning messages that can be safely ignored. Then to analyze all networks, run `./analyze_all.sh`.

The results of `./analyze_all.sh` can be imported into Matlab for further processing. Specifically: the files `G...clustering-...csv` have one line per node with three columns: node ID, degree and local clustering coefficient. The files `G...degree-...csv` contains the degree distribution (first and second column) and the cumulative degree distribution (first and third column).

Task 2: Structural Balance

Use SNAP to study the who-trust-whom online social network of the consumer review site `Epinions.com`.[1] Specifically, investigate whether structural balance in this network seems to hold, by sampling triangles at random, and checking whether they are balanced or not.

The network is directed but you may ignore the directions of the links. A small fraction of the links will have inconsistent signs (+ in one direction, − in the other) – we ignore those links.

To sample triangles at random and look at the signs of their edges, you may use the program `structbal.py`.[2] (In a terminal window, run `python structbal.py`. Some versions of the SNAP library have a memory management bug [that I have reported!] causing the program to terminate with an error message, however, you can ignore this.)

Questions to answer:

- How large fraction of the links, approximately, are positive and negative, respectively?
- From a quick inspection, does structural balance hold in this network?
- Go through the program `structbal.py` and explain how it works, line by line.

Task 3: Neighborhood Overlap

The purpose of this task is to examine the correlation between tie strength and neighborhood overlap, using a real YouTube social network. The specific goal is to produce a graph similar to Figure 3.7 in [1] that shows the correlation between these two metrics.

We use real, anonymized data consisting of a fragment of the YouTube social network, collected by SNAP. The data consist of an undirected network of friends, where each node is a YouTube user, and an edge exists between two users if they are friends. Furthermore, each user may be a member of several different smaller communities, or "groups".

[1]From the SNAP dataset description: This is a who-trust-whom online social network of a general consumer review site `Epinions.com`. Members of the site can decide whether to "trust" each other. All the trust relationships interact and form the Web of Trust which is then combined with review ratings to determine which reviews are shown to the user.

[2]The sampling of triangles performed by this program is not *uniformly* random, but that is not important here.

The task is to find out if there is a correlation between neighborhood overlap of two users and the tie strength of the edge connecting these two users. The neighborhood overlap is defined in [1, Section 3.3]. We choose to represent the tie strength between two users by the number of groups that both user A and user B are members in. To plot the figure, you will have to find the tie strength and neighborhood overlap for all edges in your network. Your plot should, as Figure 3.7 in [1], have neighborhood overlap on the vertical axis and the cumulative percentage of tie strength on the horizontal axis.

This task will require programming on your own, in Matlab, Python, or similar language.

Questions to answer:

- Discuss the visual appearance of the resulting plot, in particular in relation to the expectations from the example in [1].
- Explain how your program works.

Hints for Task 3

- Details of the data file format are in the file `README-students.txt`
- Each (connected) pair of nodes will give one value of tie strength and one neighborhood overlap. In order to see any correlation you need to calculate the *average* neighborhood overlap for a given tie strength.
- Some useful tips for handling large datasets (useful also for other labs):

  1. First load the data into Matlab:

     ```
     >> mutual_friends = load('mutual-friends.txt')
     ```

  2. Then create a sparse matrix. The easiest way to do this is to use the Matlab function sparse(i,j,s,m,n), where

     – i = vector that contains the destination nodes (the "to" nodes) in a directed network
     – j = vector that contains the source nodes (the "from" nodes)
     – s = vector that contains the desired values for element (i,j) in the sparse matrix. This would be all 1s in our case, since we are dealing with unweighted networks.
     – m,n = the desired dimensions of the sparse matrix. This would be the number of nodes in out network, for both dimensions (a square matrix).

     [Note that in an adjacency matrix, (i,j) = 1 means that there is a link from node j to node i. Hence the definitions of vectors i and j above.]

     Thus, we can do like this:

```
>> nr_nodes = max([mutual_friends(:,1); mutual_friends(:,2)]);
```

This is the value for m,n

```
>> nr_edges = length(mutual_friends(:,1));
```

This is how long the vector s should be (since each edge corresponds to a 1-element in the sparse matrix)

```
>> adj_mat_directed = sparse(mutual_friends(:,2),
     mutual_friends(:,1), ones(nr_edges, 1), nr_nodes, nr_nodes);
```

[Note the order of columns 2 and 1 of mutual_friends in the first two arguments!]

3. Make sure that the matrix is symmetric (since we are dealing with an undirected network): Note that "adj_mat_directed" is the adjacency matrix for a directed network. To obtain the symmetric version (i.e. for an undirected network), just add the matrix to its own transpose:

   ```
   >> adj_mat = adj_mat_directed + adj_mat_directed';
   ```

   Now we can manipulate adj_mat as any old (sparse) adjacency matrix. Note however that operations on elements/rows/columns in a loop (e.g. a for-loop) are rather slow since Matlab needs to unpack the sparse matrix every time we modify its contents. But matrix multiplication and other operations that do not change the actual contents are optimized for sparse matrices.

   Some final points:

   – What happens if we instead just write sparse(mutual_friends)? Well, note that mutual_friends is an k-by-2 matrix in its own right, where k is the number of edges in the edge list. But sparse(mutual_friends), without the extra arguments we used above, expects mutual_friends to be the full matrix that we are interested in. It's not; it's an edge list! This will create a much bigger matrix than expected, with the wrong contents.

   – Think the problem through – are we dealing with an directed or an undirected network? If it is directed, execute steps 1 and 2 only. If it is undirected, execute step 3 as well. For undirected networks, an edge list contains only the edge once (typically sorted after the source nodes).

   – Keep in mind that (i,j) = 1 means that there is an edge from node j to node i.

Examination

- The program code you have written should be submitted to Urkund: `ollab13.liu@analys.urkund.se`.

- Collaboration on this homework in small groups is encouraged, but each student should perform programming work individually, and individually demonstrate understanding of all tasks.
- Library functions from SNAP, Matlab and the Python standard library may be used freely, but copying of code from other libraries and/or toolboxes, from the Internet, from other students, or from previous years' students is prohibited.
- Individual oral examination takes place in class (computer lab).

References

[1] D. Easley and J. Kleinberg, *Networks, Crowds and Markets*. Cambridge University Press, 2010.