Community Detection with GANXiSw

Jierui Xie, Boleslaw K. Szymanski
(Last updated September 29, 2012)

Thank you for using our GANXiSw (also known as SLPA/SLPAw) algorithm. GANXiSw can be used for disjoint and overlapping community detection and works on weighted/unweighted and directed/undirected networks. The following is a very crude instruction for getting GANXiSw running. With input from users, we continue adding new features to make GANXiSw better. Should you have any questions or comments, please send emails to jierui.xie@gmail.com or szymansk@cs.rpi.edu.

For more information about this algorithm, please refer to our papers [1,2]. The performance of this algorithm is also discussed in survey [3].

Download: `https://sites.google.com/site/communitydetectionslpa/`

1 Important notes to existing users

What is new:

NOTE 1: Since v3.0.2, we merge both GANXiS (for unweighted/undirected networks only) and GANXiSw into only a single program GANXiSw, which works on weighted/unweighted and directed/undirected networks. GANXiSw provides many more parameters for advanced users. You are recommended to use GANXiSw starting from v3.0.2 (Java). (C++ version is under development)

NOTE 2: GANXiSw considers every edge to be *directional* explicitly (a general case for an edge). That means in an undirected network, for each edge i-j, there are two edges i→j and j→i in its corresponding weighted network. If you are using an undirected network and in your input file you already have both i→j and j→i, NO NEED to change anything in the command line. However, if you are not sure, you can add parameter **-Sym 1** to ensure a UNdirected networks at runtime.

Changes and new features

- Allow to read node names or ids of any type, numeric or text string (see 3).
- Parameter -Sym to ensure bi-directional edges (or undirected edges) (see 3.1).
- Parameter -seed allows to reproduce results (see 8.2).
- Parameter -W allows to treat a weighted network as a unweighted network at runtime (see 8.1).
- Parameter -loopfactor allows to control the running time when decomposing large communities (see 8.5.1).

2 Java Users

You need Java (JRE or JDK) with a version 1.5 or higher to run the program.

3 Input format for directed/undirected networks

GANXiSw accepts a file containing a list of DIRECTED edges. Each row represents an edge. It is a triple (node1 node2 weight), meaning there is an edge from node1 to node2. The name or ID for a node could be any number or string. The value for a weight could be any positive real number (e.g., 0.5, 2, 5.43, 100.0). If weights are not provided in the third column, they are automatically assigned a value 1.0. Each column is separated by spaces or tabs. The following shows examples of accepted edges:

AA BB
1 2
...

or
AA BB 1.23
1 2 100
...

See example network test.ipairs or test_str.ipairs for an example.

3.1 Note for UNdirected networks

Each edge is treated as a directed edge. So, (node1 node2 1.5) means there is an edges from node1 to node2. To treat the input as an undirected network, you can simply set parameter **-Sym 1**. GANXiSw will automatically create two edges for you, (node1 node2 1.5) and (node2 node1 1.5).

4 Minimum parameters to know

- Use **-i** to specify the input network.
- Use **-Sym** to ensure an undirected network.
- Use **-help** to list all parameters.

Note that parameters are *CASE-SENSITIVE*, e.g., -Onc is not -onc.

The simplest command for directed networks is
java -jar GANXiSw.jar -i networkfile

The simplest command for undirected networks is

java -jar GANXiSw.jar -i networkfile -Sym 1

For help and list all parameters
java -jar GANXiSw.jar -help or
java -jar GANXiSw.jar

5 Output format

By default all the output files are put in the output directory. Each file (.icpm) is a cover/partitioning found by GANXiSw. Each line contains nodes in a community.

There are other options for output formats.

By setting parameter **Onc** to 1, you can output communities in anther form other than .icpm. A file with a name ending with *.node-com.txt* is generated. Each row contains a pair of <nodeID communityID>.

java -jar GANXiSw.jar -i test.ipairs -Onc 1

Similarly, by setting parameter **Onc** to 2, a file with a name ending with *.com-node.txt* is generated. Each row contains a pair of <communityID nodeID>.

java -jar GANXiSw.jar -i test.ipairs -Onc 2

6 How to perform disjoint community detection with GANXiSw

The minimum command for disjoint community detection is to add parameter **-ov 0** as follows:

java -jar GANXiSw.jar -i networkfile -ov 0

Try it on an example network:
java -jar GANXiSw.jar -i test.ipairs -ov 0

You should see an output file called *GANXiSw_test_run1_r0.5.icpm* under the folder *output*. (The reason for r0.5 is because setting -ov 0 is equivalent to setting -r 0.5, see papers [1,2]).

Most parameters in the following section apply to disjoint detection as well.

7 How to perform overlapping community detection with GANXiSw

Simply type:

java -jar GANXiSw.jar -i networkfile

This is the minimum command to run GANXiSw for overlapping detection. Try the following to see if you can run the GANXiSw:

java -jar GANXiSw.jar -i test.ipairs

7.1 With a specified threshold

By default, GANXiSw is set for overlapping community detection. It runs once with ten different thresholds, including $r \in \{0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5\}$ (for r, see papers [1,2]). Therefore, you will see ten output files when you run with the above minimum command. You can provide a specified threshold (e.g., 0.45) as in the following example:

java -jar GANXiSw.jar -i test.ipairs -r 0.45

7.2 Repeat many times

Since GANXiSw is non-deterministic process, usually you want to repeat several times and either record the best performance or take the average performance. You can specify the number of times to repeat. The following command asks GANXiSw to repeat each threshold 3 times

java -jar GANXiSw.jar -i test.ipairs -run 3

7.3 Set the maximum iteration

GANXiSw is an iterative process, you can change the maximum number (e.g., 50) of iteration as follows:

java -jar GANXiSw.jar -i test.ipairs -t 50

7.4 Change the output directory

The default output directory is the output. You can change it to other as follows:

java -jar GANXiSw.jar -i test.ipairs -d myOutputDirectory

7.5 Use only the largest component

By setting L to 1, you can ask GANXiSw to use only the largest connected component in the network.

java -jar GANXiSw.jar -i test.ipairs -L 1

7.6 Output overlapping nodes

By setting parameter **Oov** to 1, you can output a list of overlapping nodes (if any) in a file with a name ending with *.ovnodes.txt*.

java -jar GANXiSw.jar -i test.ipairs -Oov 1

7.7 Explore the listener's memory

By setting OMem1 to 1, you can ask GANXiSw to output the labels stored in each node's memory. An output file with a name ending as L1.mem.txt is generated. Each line contains < nodeID, Label1, Label2, ... Labelt>. There is a total of t (maximum iteration) labels.

java -jar GANXiSw.jar -i test.ipairs -OMem1 1

8 Advanced parameters

Advanced parameters are introduced allowing users to have more control on the input/output. This might be particularly useful for some applications (e.g., biology). The value for a weight could be any positive real number. The edge weights of a node is normalized locally.

8.1 Treat a weighted network as an unweighted network

Given a weighted network, you might want to treat it as an UNweighted network at runtime. By doing so, weights are ignored. One example of using this parameter is that you might want to see the effect of the weights (by comparing networks with/without weights).

To run your network without weights, set parameter **-W** 0.

java -jar GANXiSw.jar -i test.ipairs -W 0

For comparison, you might need to set the **-seed** parameter for runs with/without weights (see 8.2).

8.2 To reproduce results with a seed

By default, a random seed is used for each run. To reproduce your results, you can specify a seed, which is a long data type. You can also compare the effects of different parameter settings by keeping the same seed. How to get a seed? After each run, GANXiSw prints out the seed used. You can save this seed and use for parameter **-seed** for other runs. The following examples show how to compare results with/without weights (see 8.1).

java -jar GANXiSw.jar -i test.ipairs -seed 7733972961972440000

java -jar GANXiSw.jar -i test.ipairs -W 0 -seed 7733972961972440000

8.3 Select speaking and listening rules

GANXiSw implements different rules for interactions, which might result in slightly different output. You can set parameter **v** to 1, 2 or 3 (default value) to make a choice, e.g.,

java -jar GANXiSw.jar -i test.ipairs -v 3

8.4 Control the minimum output community size

By default, GANXiSw tries to output communities with size at least **minC**=2. You can set parameter minC to other positive values for your need. Note that any isolated community with size smaller than minC remains in the output.

java -jar GANXiSw.jar -i test.ipairs -ov 0 -minC 10

8.5 Control the maximum output community size

GANXiSw allows you to limit the size of a community such that it is smaller than **maxC**. By default, this parameter is inactive. To activate it, you can set it to any positive value larger than minC.

java -jar GANXiSw.jar -i test.ipairs -ov 0 -maxC 15

java -jar GANXiSw.jar -i test.ipairs -ov 0 -minC 5 -maxC 15

Communities with size above maxC are broken down by a series of embedded GANXiSw. The speaker and listener rule of the embedded GANXiSw is determined by parameter **ev**, which is similar to the definition in section 8.3. The default value is ev=1.

java -jar GANXiSw.jar -i test.ipairs -ov 0 -minC 5 -maxC 15 -ev 1

You might want to try ev=2 if you have difficulty in breaking down the communities, which usually has a better chance to be successful. Note that, setting maxC does not guarantee that all output communities have size larger than this value.

java -jar GANXiSw.jar -i test.ipairs -ov 0 -minC 5 -maxC 15 -ev 2

8.5.1 Special note for using maxC and running time

When using the -maxC, the parameter **-loopfactor** is used to control how many tries to decompose the communities with size beyond your specified value (maxC). It is a multiply factor. The actual number of tries is loopfactor times number of large communities found. By default, it is 1.0. It is very important to reset this value for large networks.

If you observe that the GANXiSw takes too long on decomposition, you should restart the program and set **loopfactor** to a smaller value, e.g., 0.2.

If GANXiSw finishes but output many communities with size beyond what you want, you can increase the number of tries by set this parameter to a larger value, e.g. 5.0. However, it might require much long time.

In short, the parameter will effect the running time, the larger, the longer running time.

java -jar GANXiSw.jar -i test.ipairs -maxC 10 -loopfactor 0.8

8.6 Output the label distribution for each node

Parameter **Ohis1** provides a way to output the label distribution in each node's memory. Two files together are generated to describe the distributions by setting Ohis1 to 1.

java -jar GANXiSw.jar -i test.ipairs -ov 0 -Ohis1 1

java -jar GANXiSw.jar -i test.ipairs -ov 0 -minC 5 -maxC 15 -Ohis1 1

The file with a name ending with *L1.histlabel.txt* contains labels with the largest five probabilities in a distribution. Each row contains a node ID, followed by top five labels. For example (1 3 4 1 -1 -1), this row is for node ID=1, the most frequent label is 3, followed by 4 and 1. A label -1 indicates N/A.

The file with a name ending with *L1.histprob.txt* is generated containing the corresponding probabilities for top five labels. For example (1 0.64 0.35 0.01 0 0), this row is for node ID=1, the max probability is 0.64. A probability 0 indicates N/A.

If the parameter maxC is used (as in section 8.5), then you can set parameter **Ohis2** to 1 to output the final distribution (after broken down). Two more files *L2.histlabel.txt* and *L2.histprob.txt* are generated accordingly.

java -jar GANXiSw.jar -i test.ipairs -ov 0 -maxC 15 -Ohis1 1 -Ohis2 1

java -jar GANXiSw.jar -i test.ipairs -ov 0 -minC 5 -maxC 15 -Ohis1 1 -Ohis2 1

9 Copyright and Disclaimer Notice

of any current or pending patent rights. The authors of the software make no representations about the suitability of this software for any particular purpose. The entire risk as to the quality and performance of the software is with the user. Should the software prove defective, the user assumes the cost of all necessary servicing, repair or correction. In particular, neither Rensselaer Polytechnic Institute, nor the authors of the software are liable for any indirect, special, consequential, or incidental damages related to the software, to the maximum extent the law permits.

10 References

Algorithm and Survey:

[1] Jierui Xie, Boleslaw K. Szymanski and Xiaoming Liu, *SLPA: Uncovering Overlapping Communities in Social Networks via A Speaker-listener Interaction Dynamic Process*, IEEE ICDM workshop on DMCCI 2011.

[2] Jierui Xie and Boleslaw K. Szymanski, *Towards Linear Time Overlapping Community Detection in Social Networks*, 16th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), 2012.

[3] Jierui Xie, Stephen Kelley and Boleslaw K. Szymanski, *Overlapping Community Detection in Networks: the State of the Art and Comparative Study*, ACM Computing Surveys 45(12) December 2013.