User's guide for OSLOM (version 2.5)

October 2, 2013

Thanks for downloading the code which implements OSLOM.

1 Compiling

The program comes along with a file called **compile_all.sh**.

Type

./compile_all.sh

from a Unix (MAC) terminal. If you are using Windows, you could still run the program by installing MinGW (Minimalist GNU for Windows, `http://www.mingw.org/`). If you see something like **./compile_all.sh: Permission denied**, type:

chmod 744 compile_all.sh

which makes the script executable and try again with:

./compile_all.sh.

2 What you get

After compiling, you should see a number of binary files. **oslom_undir** and **oslom_dir** implement OSLOM for undirected and directed networks. The option **-f** introduces the name of the file where the network is stored. Try:

./oslom_undir -f example.dat -uw

and when the program is done (it takes about 15 seconds on a laptop computer), type:

./pajek_write_undir example.dat

Now look at the folder **example.dat_oslo_files**. You should see a few files, described in the coming subsections.

2.1 The modules

The most important files are **tp** and **tp1** (**tp** is also copied in the main folder, for your convenience).

In **tp** you find the set of (overlapping) modules at the lowest hierarchical level. The file content looks like this:

#module 0 size: 18 bs: $5.69337e{-}101$
7 10 13 17 18 22 28 31 33 37 38 41 44 45 47 50 51 56

The line starting with # says that the module with id 0 has 18 nodes and score of $5.69337e{-}101$, which is an estimation of the probability of finding a module like this one in a random network (very very low in this case). To be specific, this is in fact an upper bound of the real significance. In any case, the program stops when the score is below the $p-$value (note that it can be much much lower). The second line gives the nodes in the module.

You should also see something like:
#module 6 size: 29 bs: 0.756234
1 2 3 4 9 15 21 23 27 30 34 43 46 48 49 53 60 63 68 70 72 73 74 78 79 81 83 93 1000

The score is higher than the $p-$value because the module contains a node which is non-significant. Running the program with option **-singlet** will tell that node 1000 has not been assigned to any module, i.e. it forms a module on its own. In other words, it is a homeless node. We recommend to use this option only if you are interested in detecting noisy nodes.

tp1 is the analogous for the first hierarchical level. In our example, this is all, there are no other hierarchical levels for the network stored in *example.dat*. In general, if there are further levels, the program will produce other files: **tp2**, **tp3**, etc..

2.2 The pajek files

Now open the file called **pajek_file_0.net**. The format of this file is the usual pajek format and you can use **Pajek** (`http://vlado.fmf.uni-lj.si/pub/networks/pajek/`) and/or **Gephi** (`http://gephi.org/`) to visualize the network. Each module of the lowest level has a different color. Overlapping nodes are black, homeless nodes are white. The only difference with **pajek_file_1.net** is that it has different colors to highlight higher level modules.

If the visualization looks too small, try:
./pajek_write_undir example.dat 2.5
You would get another pajek file which would produce a picture twice and a half times bigger. Type the command again, to have different visualizations (colors and positions are chosen through a stochastic algorithm).

N. B.: All the output files are stored in the directory called **[network file]_oslo_files**. If the directory is not empty it will be cleared, so be careful if you want to save some previous output files.

2.3 What else?

This is not fundamental, so if you are impatient, skip this and go to Section 3.1.

Here is a list of the other output files:

1. **net1** contains the edge list of the network of communities found at the lowest hierarchical level. For instance,

   0 10 5

   means that there are five links between node 0 and node 10. The labels 0 and 10 are the module ids of file **tp**. Likewise, file **net2** stores the network of communities found at the second hierarchical level, whose labels are those used for the modules in file **tp1**, etc.. In the case of weighted networks, you would see four numbers like these:

   0 10 15.76 5

   which means that there are five links between node 0 and 10 and the sum of the weights of these links is 15.76.

2. **short_tp1** (**short_tp2**) gives the modules of the community network **net1** (**net2**, **net3**, ...) using its labels, which refer to the ids of the modules listed in **tp** (**tp1**, **tp2**, ...).

3. **partitions_level_0(1)** contains the pruned modules the program found at the end of each run.

4. **statistics_level_0(1)** contains some basic statistics of the modules found for each hierarchical level.

5. **pos_0** is written by **pajek_write_undir**. It gives the position of the nodes of **example.dat**, the format is **x y node**. **pos_1** is the analogous for the nodes in **net1**, **pos_2** for the nodes in **net2**, etc..

3 OSLOM's Options

3.1 Basic Options

1. **Directed vs Undirected**. First of all, you need to choose between

   ./oslom_undir

   and

   ./oslom_dir.

As you can easily guess, the former is for the undirected networks, the latter for directed ones. The same holds for **pajek_write_undir** and **pajek_write_dir**. If you have a directed network, the format of the input file is **node1 node2**, meaning that there is a link from **node1** to **node2**. Self loops and repetitions are ignored in both cases.

2. **-uw (-w)** is the only mandatory flag.

   Option **-uw** is for unweighted networks, or better it is for weighted networks where considering weights as multiple links. This means that it is possible to use a format for the network file like:

   node1 node2

   or

   node1 node2 m_{12}

   but m_{12} must be an integer number! It tells how many links are between the two nodes. In the null model, these links are randomly rewired along with all the links of the network.

   Option **-w** is to choose the weighted version of the algorithm. This selects a different null model where the strength of a node is shared between its neighbors. The input file can be:

   node1 node2 w_{12}

   or

   node1 node2 w_{12} m_{12}

   where w_{12} is the weight between **node1** and **node2**. If there are m_{12} multiple links between the two nodes, w_{12} must be the sum of the weights on those links.

   About the labels of the nodes, they must be non-negative integer numbers, but they do not have to be consecutive and can start from any number.

3.2 Other Options

1. **-r** m sets equal to m the number of runs for the first hierarchical level. The default value is 10. This value depends on your patience. The bigger m, the more accurate the results. We are working to let the program choose the optimal number of runs according to the convergence of the results. The option **-r 0** has to be employed when OSLOM starts from a partition found by other algorithms.

2. **-hr** m sets the number of runs for higher hierarchical level equal to m. The default value is 50 (the method should be faster since the aggregated network is usually much smaller). If you are not interested in hierarchies set **-hr 0** and the method will stop after finding the lowest level partition.

3. **-seed** m. The program is not deterministic, so it needs a seed for its random number generator. Using this option it will be set equal to m, otherwise the seed will be set reading the file **time_seed.dat**, which will be updated afterwards.

4. **-hint filename**: reads a file with a partition previously found by another algorithm. The file has to be formatted like **tp** (lines starting with # are skipped, so they are optional): it simply has to contain the modules separated by an end of line. All the modules read from the file will be cleaned up by the algorithm, which would retain only the significant ones. Then, the program would look for submodules and cleanup all of them. The outcoming clusters will be written in file **partitions_level_0**, together with the modules found during the cluster search (unless you set **-r 0**). This option can be used only for the first hierarchical level.

5. **-load filename**: reads a file with a partition previously found by OSLOM. The file must have the format of the output file **tp** (see Section 2.1). In particular, this option can be useful if you want to parallelize the program. You have to run OSLOM in different folders with different seeds (option **-seed**).

6. **-t** l sets the $p-$value equal to l. The default value is 0.10. Interestingly, if you increase this value you get more modules. The reason is that submodules are more easily considered significant.

7. **-singlet**: singletons. The program usually finds a number of nodes which are not assigned to any module. By default, the program assigns "homeless" nodes to a module according to the score of the node with respect to the existing modules: the module with best score will "accept" the node (this only applies to the lowest hierarchical level). Setting this flag, instead, homeless nodes are not assigned to any cluster. Also, a file called **tp_without_singletons** will be written with all the nodes assigned (as by default). This option should be used if you are interested in filtering nosily nodes (typically low-degree nodes).

8. **-cp** l sets the *coverage parameter* equal to l. This parameter is a kind of resolution parameter: it is used to decide between taking some modules or their union. Default value is 0.5. Bigger value leads to bigger clusters. l must be in the interval $(0, 1)$.

9. **-fast**: this flag is to have fast results. It is equivalent to set **-r** 1 **-hr** 1, so it enables the fastest possible execution of the program.

10. **-infomap** r: the program will call another program (**infomap**) and will apply OSLOM's cleanup procedure to the modules found by it. r is the number of times **infomap** will be called, good values are between 1 and 10.

11. **-copra** r, **-louvain** r are similar flags for other programs. They can be used simultaneously. The general idea is that the more programs are used, the better the exploration of possible modules will be, but of course more time is needed.

Examples:

./oslom_undir -f example.dat -uw -infomap 3 -copra 2 -louvain 1 -r 2

This will run oslom on example.dat using our method to explore the graph (two runs) and the outputs of infomap after 3 iterations, of copra after 2 and of louvain as initial conditions [Oslom cleans-up afterwards the found modules]. The final output is formed by the best modules found with all these methods.

./oslom_undir -f example.dat -uw -infomap 1 -r 0 -hr 0

OSLOM runs on example.dat using as initial modules the output of infomap. Since r = 0 and hr = 0, our module search technique is not applied.

N. B.: All the external programs have been distributed because they are free. However, if you use one or more of them for your research, please cite the paper where the method is described.

4 References

If you used this program for your research, please cite this paper: (to appear)

In addition, please cite the following papers if you used **-copra**, **-louvain** or **-infomap**.

1. if you used **-copra**, please cite: Steve Gregory, New J. Phys. **12**, 103018 (2010).
2. if you used **-infomap**, please cite: M. Rosvall and C. T. Bergstrom, Proc. Natl. Acad. Sci. U.S.A **105**, 1118 (2008).
3. if you used **-louvain**, please cite: V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech. **P10008** (2008).

Further information and the source codes can be found in the following urls:

1. **copra**: `http://www.cs.bris.ac.uk/~steve/networks/software/copra.html`
2. **infomap**: `http://www.tp.umu.se/~rosvall/code.html`
3. **louvain**: `http://sites.google.com/site/findcommunities/`