COPRA User's Guide

Steve Gregory

v1.24 – February 12, 2010

What the COPRA program does

The program reads a network from a file and computes a *cover* — a division of the vertices into a number of possibly overlapping *communities*. The amount of overlap can be controlled by the *v* parameter, which is the only parameter of the COPRA algorithm.

Network file format

The input network file must be in "list of edges" format. Each line contains one edge: two vertex names followed by an optional numeric weight, all separated by *one* space or tab character.

Blank lines and lines beginning with "#" are ignored.

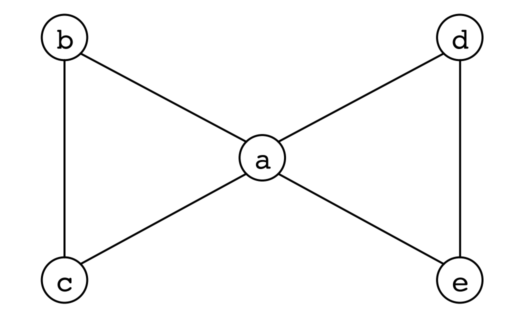
A vertex name can be any sequence of characters excluding spaces.

A weight can be any floating point number. The weight can be omitted from all or some edges, in which case the default weight (1) is assumed.

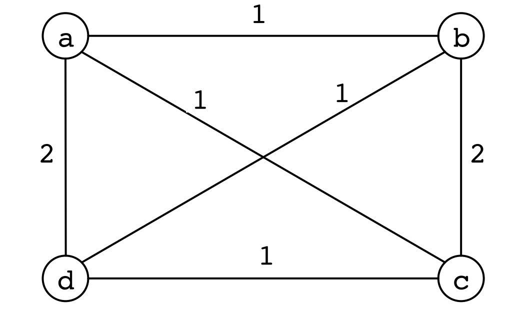
For a unipartite network, edges can be listed in only one direction or both. Self-edges are ignored, but with a warning message.

For a bipartite network, the first and second vertex names are mode-1 and mode-2, respectively, and weights are ignored. Mode-1 and mode-2 vertices are considered different even if they have the same names.

For example, the networks below can be represented as shown:



```
# Network 1
a b
a c
a d
a e
b c
d e
```



```
# Network 2
a b 1
a c 1
a d 2
b c 2
b d 1.0
c d
```

How to run the program

Download the file "copra.jar" and prepare your network in a text file; e.g., "karate.txt". You can cluster it by the command:

```
java -cp copra.jar COPRA karate.txt
```

"copra.jar" and the network file may be in any location. If they are not in the current directory, just give the appropriate pathnames instead.

The output displayed on the screen is in two parts:

1. A description of the network and the options selected.
2. Statistics about the solution (cover) produced.

The solution itself is placed in a file whose name begins with "clusters-"; e.g., "clusters-karate.txt". Each line of this file contains one community: a set of vertex names separated by spaces.

It is sometimes useful to run the algorithm several times and keep the solution with the maximum modularity. To do this, use the "-mo" and "-repeat" options:

```
java -cp copra.jar COPRA karate.txt -mo -repeat 10
```

This shows a new section of output on the screen:

3. Averages of statistics about the solutions produced, as well as the standard deviation and maximum modularity.

It also keeps the best (maximum modularity) solution, in a file whose name begins with "best-clusters-"; e.g., "best-clusters-karate.txt".

Different files are generated when clustering a bipartite network. For example:

```
java -cp copra.jar COPRA southernwomen.txt -bi
```

The solution is again placed in a file named "clusters-southernwomen.txt", but now each line of the file contains two communities: the mode-1 community and the mode-2 community, separated by a tab character. Additionally there are two extra files: "clusters1-southernwomen.txt" and "clusters2-southernwomen.txt", which contain the mode-1 communities and mode-2 communities, respectively.

You can also run the algorithm repeatedly and keep the solution with the highest modularity, as for a unipartite network. E.g.:

```
java -cp copra.jar COPRA southernwomen.txt -bi -mo -repeat 10
```

but there are a few points to note:

1. The modularity is computed for each mode separately, using the usual unipartite modularity measure. The modularity of the cover contained in "clusters1-southernwomen.txt" (resp. "clusters2-southernwomen.txt") is computed with respect to the corresponding unipartite projection in "southernwomen-1.txt" (resp. "southernwomen-2.txt"). If these projection files do not exist, you can create them using the Project command below.
2. The "best" solution stored is the one with the maximum modularity for the mode-1 cover. This might not be the one with the best mode-2 cover.
3. The "best" solution is kept in a file, named "best-clusters-southernwomen.txt", that contains both modes.

Note: If the network is too large, a runtime error may occur; to solve this, use Java's `-Xmx` option to increase the maximum heap size, but do not exceed the physical memory available.

Command syntax and options

In general, to run the COPRA program:

```
java -cp copra.jar COPRA filename [options]
```

where `options` include:

`-bi`

filename is a bipartite network. Each edge listed in the file contains a mode-1 vertex name followed by a mode-2 vertex name, followed possibly by a weight (which will be ignored). Mode-1 and mode-2 vertices are disjoint even if they have the same names. E.g., an edge {a,a} is allowed and is not a self-edge. This option may not be used in conjunction with the "`-w`" option.
Default: the network is unipartite; self-edges will be ignored, with a warning message.

`-w`

filename is a weighted network. If there are actually no weights in the file, or if all edges have the same weight (the default weight of an edge is 1), a warning message will be printed. Otherwise, the weights will be used in the clustering process and may affect the result. This option may not be used in conjunction with the "`-bi`" option.
Default: the network is unweighted; edge weights may still appear in the file but will be ignored.

`-v` *v*

Sets the *v* parameter (the maximum number of communities per vertex) of the COPRA algorithm to *v*.
Default: *v*=1.

`-vs` v_1 v_2

Executes the COPRA algorithm with the *v* parameter set to v_1, v_1+1, ..., v_2. If used in conjunction with the "`-repeat`" option, the execution is performed for each value of *v*.

`-prop` *p*

Limits the maximum number of iterations to *p*. The propagation will end as soon as the termination condition is satisfied, even if this happens after less than *p* iterations.
Default: no limit, so the propagation will continue until the termination condition is satisfied.

`-repeat` *r*

Repeats the execution *r* times. If used in conjunction with the "`-vs`" option, for each value of *v*, the execution will be repeated *r* times. If *r*>1, the screen display will show the averages of the statistics, as well as the standard deviation and maximum of the modularity.
Default: *r*=1.

`-mo`

Compute the overlap modularity (Nicosia et al.) of each solution. It is displayed on the screen, and used to decide which is the best solution to keep when "-repeat" is used. This is optional because the overlap modularity measure can be expensive to compute for large networks.

`-nosplit`

Do not split discontiguous communities into contiguous subsets. If this option is selected, the "communities" in the solution will not be valid communities.

`-extrasimplify`

Simplify the solution (by removing communities that are subsets of others) again after splitting discontiguous communities. This is always done anyway before the splitting, but splitting can create new communities which are subsumed by others. By default these are not removed, but they are when this option is selected. This operation is optional because its complexity is $O(n^2)$, but in practice it is usually very fast and worthwhile.

`-q`

Do not display the description of the parameters and the network when the program starts.

Other useful commands

To project a bipartite network onto each of its two modes, use the command:

```
java -cp copra.jar Project southernwomen.txt
```

which puts the mode-1 and mode-2 projections into files "`southernwomen-1.txt`" and "`southernwomen-2.txt`", respectively.

The "`-w`" option can be used to create a weighted projection. You can use the "`-s`" option to add a suffix to the projection filenames, in order to keep weighted and unweighted projections separate. For example, to put mode-1 and mode-2 weighted projections into files "`southernwomen-1w.txt`" and "`southernwomen-2w.txt`", use:

```
java -cp copra.jar Project southernwomen.txt -w -s w
```

To compute the overlap modularity of a cover with respect to a network, use:

```
java -cp copra.jar ModOverlap karate.txt clusters-karate.txt
```