# CONGA User's Guide

Steve Gregory

August 31, 2007

## What the CONGA program does

The program reads an undirected, unweighted graph (with *n* vertices and *m* edges) from a file and computes a *clustering* — a division of the vertices into *k* possibly overlapping *clusters* — for all values of *k* from 1 to at least *n*. This clustering information is stored in a file, to allow you to view the clustering for any value of *k* without recomputing it each time.

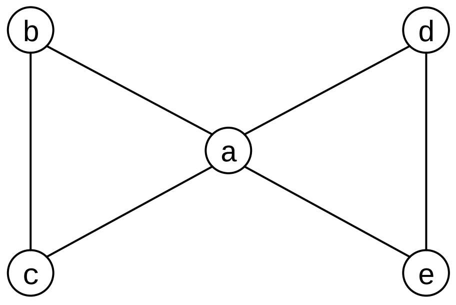
## Graph file format

Two alternative formats are accepted for the input graph file:

- **CONGA format**. A line containing the name of each vertex, each of which is followed by a line for each of its neighbours, beginning with "`--`" and a space. Edges must be listed in both directions; if not, the file is rejected.
- **CFinder format**. A line for each edge: a pair of vertex names separated by a space or tab. Edges can be listed in only one direction or both.

In both formats, self-edges are ignored. Blank lines and lines beginning with "#" are also ignored.

For example, the graph below left can be represented in two formats as shown below right:



| **CONGA format** | **CFinder format** |
|---|---|
| `# Graph 2` | `# Graph 2` |
| `a` | `a b` |
| `-- b` | `a c` |
| `-- c` | `a d` |
| `-- d` | `a e` |
| `-- e` | `b c` |
| `b` | `d e` |
| `-- a` | |
| `-- c` | |
| `c` | |
| `-- a` | |
| `-- b` | |
| `d` | |
| `-- a` | |
| `-- e` | |
| `e` | |
| `-- a` | |
| `-- d` | |

## How to run the program

Download the file "`conga.jar`" and prepare your graph in a text file; e.g., "`karate.txt`". Assuming this is in CONGA graph format, you can do the clustering by the command:

```
java -cp conga.jar CONGA karate.txt
```

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

If your graph file is in CFinder format (a list of edges), add the "`-e`" option; e.g.:

```
java -cp conga.jar CONGA edolphins.txt –e
```

The output displayed on the screen is in up to four parts:

1. An explanation of the options selected.
2. **Finding clusters**. This shows the steps in the clustering process.
3. **Results**.
4. **Statistics**.

The above commands produce no results, so the "Results" section is empty, and the "Statistics" section give statistics only about the original graph and the clustering process.

Whenever a graph is clustered, a file is produced with a name beginning "`clustering-`" (e.g., "`clustering-karate.txt`") that contains the clustering information. If you run one of the above commands for a second time, the clustering will not be performed again, and the "Finding clusters" and "Statistics" sections will be empty. If you want to force the program to recompute clusters (e.g., because the graph has changed), use the "`-r`" option.

One way to get results is using the "`-m`" option:

```
java -cp conga.jar CONGA edolphins.txt -e -m
```

This shows, in the "Results" section, some statistics about every clustering:

1. **Mod**: Newman's modularity measure. This is well-defined for the GN algorithm, in which clusters do not overlap, but is meaningless for the CONGA algorithm.
2. **NewMod**: I forget what this is!
3. **Vad**: *Vertex average degree* (the number of intracluster edges of each vertex, averaged over all vertices).
4. **Overlap**: The sum of the sizes of all clusters divided by *n*, the number of vertices in the graph.

Another way is to use the "`-n`" option to view the clusters in a particular clustering. For example, to show the division into two clusters

```
java -cp conga.jar CONGA edolphins.txt -e –n 2
```

This shows, in the "Results" section, the vertices in each of the clusters. Each cluster is prefixed by its size, followed by "`:`".

For very large graphs, it is sometimes useful to view only some of the clusters. You can use the "`-f`" option to see all clusters containing a specified vertex. E.g., to see the cluster(s) containing Donald the dolphin:

```
    java -cp conga.jar CONGA edolphins.txt -e –n 2 –f Donald
```

When the "`-n`" option is used, the cluster contents are also written to a file with a name beginning "`clusters-`" (e.g., "`clusters-edolphins.txt`"). Each cluster is shown on a separate line, with vertices separated by spaces, and no size prefix.

The "`-n`" option also displays, in the "Statistics" section, some statistics about the specified clustering. The information is the same as shown by the "`-m`" option plus the number of intracluster and intercluster edges compared with the number of intracluster and intercluster vertex pairs (i.e., the maximum possible number of intracluster and intercluster edges).

## Command syntax and options

To run the CONGA program:

```
    java –cp conga.jar CONGA <filename> [options]
```

where `options` include:

`-e`

Graph file format is a list of edges.
Default: graph file format is CONGA native format.

`-g fF`

Use file `fF` as a filter. This should be a text file with one vertex name on each line. When the graph is read in from `<filename>`, vertices appearing in `fF`, and edges to them, are omitted from the graph.
Default: no filter is used, so all vertices in `<filename>` are added to the graph.

`-n nC`

Show the clustering containing `nC` clusters, or none if `nC` = 0.
Default: `nC` = 0.

`-f v`

Show only the cluster(s) containing vertex `v`, in the specified clustering, where nC > 0.
Default: show all clusters in the clustering.

`-r`

Recompute the clustering information even if an appropriate clustering file exists.

`-m`

Show basic statistics (modularity, etc.) about every clustering with at least `nC` clusters.
Default: don’t compute or display these statistics.

`-GN`

Run the GN (Girvan and Newman) algorithm. In this algorithm, clusters never overlap.
Default: run the CONGA algorithm.

`-a aF`

If `aF` = 0, use a dynamic data structure to store the betweenness values.
If `aF` > 0, use a 2D array whose size is `aF` * *n*, where *n* is the number of vertices in the graph.
The dynamic method uses less memory but more execution time than the array method.
If the array method is specified (`aF` > 0), `aF` must also be at least 1, or the program will soon fail. For the GN algorithm, `aF` = 1 is always sufficient. For the CONGA algorithm, `aF` must be more than 1, but the size needed depends on the amount of overlap in the graph. If `aF` is set too small, a runtime error will occur when the array overflows, usually at a late stage. If `aF` is set too large, a different runtime error will occur at the beginning; to solve this, use Java’s `-Xmx` option to increase the maximum heap size, but do not exceed the physical memory available.
Default: `aF` = 0.