Manual
for the
Community Detection Toolbox v. 0.9

M. Mitalidis, Ath. Kehagias, Th. Gevezes and L. Pitsoulis

Department of Electrical and Computer Engineering

Aristotle University of Thessaloniki

February 2014

Table of Contents

1. Introduction

A good introduction to the problem of *community detection*[1] is the following passage from [Fortunato2010]:

> The modern science of networks has brought significant advances to our understanding of complex systems. One of the most relevant features of graphs representing real systems is community structure, or clustering, i.e. the organization of vertices in clusters, with many edges joining vertices of the same cluster and comparatively few edges joining vertices of different clusters. Such clusters, or communities, can be considered as fairly independent compartments of a graph, playing a similar role like, e. g., the tissues or the organs in the human body. Detecting communities is of great importance in sociology, biology and computer science, disciplines where systems are often represented as graphs.

In this document we present the *Community Detection Toolbox* (CDTB), a MATLAB toolbox which can be used to perform community detection. The CDTB contains several functions from the following categories.

1. graph generators;
2. clustering algorithms;
2. cluster number selection functions;
4. clustering evaluation functions.

Furthermore, CDTB is designed in a parametric manner so that the user can add his own functions and extensions.

The CDTB can be used in at least three ways. The user can employ the functions from the MATLAB command line; or he can write his own code, incorporating the CDTB functions; or he can use the Graphical User Interface (GUI) which automates the community detection and includes some data visualization options.

Section 2 of the manual gives "Quick Start" instructions: how to install CDTB and how to run some simple examples. Section 3 presents some basic community detection concepts and theory. Section 4 gives more details and examples about CDTB and Section 5 gives many examples on the use of GUI. In Section 6 we summarize and present our conclusions, including ways in which CDTB can be extended. Three Appendices

[1] In the following passage, as well as in the rest of this manual, "graph" is used as a synonym of "network", "cluster" as a synonym of "community", "clustering" as a synonym of "community detection".

are also included: Appendix A includes additional community detection theory; Appendix B presents some bibliography; Appendix C is the Reference for all the CDTB functions.

IMPORTANT!!! The CDTB has been tested on Windows 7 and Matlab R2012a. It should also work on Windows XP and Windows 8, and for all recent versions of Matlab. It should also work on other platforms (Linux, Mac) with the following exception: the functions **`GCGloDens`**, **`GCQLocDens`**, **`GCQDistBased`** and **`GCQNodMemb`** call Windows executable programs, so they will **not** work on other platforms. When you run scripts which invoke these functions (e.g., **`CheckAll01.m`**) on platforms other than Windows, you will get an error message; to circumvent this, simply open these scripts with a text editor and comment out the lines invoking the offending functions.

Acknowledgement. We want to thank J. Hespanha , E. le Martelot and A. Scherrer for permission to use their code in the CDTB.

Marios Mitalidis
Thanasis Kehagias
Theodoros Gevezes
Leonidas Pitsoulis

Thessaloniki, February 2014

2. Quick Start

2.1 Installation

Installation of the CDTB is extremely simple. You can download the file CDTB.zip from http://mathworks.com. When the download is complete, unzip the file in a folder of your choice. From now on we will assume that it is the folder `C:\CDTB`. When unzipped, the folder will contain several subfolders, e.g., `C:\CDTB\Algorithms`, `C:\CDTB\Auxiliary` etc. Now you are ready to go.

2.2 A Simple Demo

Start MATLAB, go to `C:\CDTB` and in the command line type

```
>> PathAdd
```

and hit `[Enter]`. This will add to the *MATLAB Path* the subdirectories which contain the MATLAB *.m files which do the actual community detection work[2]. In the command line type

```
>> CDTBDemo01
```

and hit `[Enter]`. The *MATLAB script* `CDTBDemo01.m` will run and in the command window you will get a message which says

```
The NMI metric between V0 and Vest is 1
```

NMI is the *normalized mutual information* index of partition (i.e., clustering) similarity. It takes values in the interval [0,1]. The maximum value 1 indicates maximum similarity, i.e., identity. The partitions compared are V_0, the true partition, and V_{est}, the partition estimated by a *modularity maximization algorithm*. Hence in this experiment V_{est} is identical to V_0, This can also be seen in the plot which MATLAB presented. It looks like this

[2] Note that the new path is not saved, so you must retype `PathAdd` every time you start a new MATLAB session. Or, you can go to the MATLAB menu `File/Set Path` and click the button `Save`, which will make MATLAB memorize the path for all future sessions.

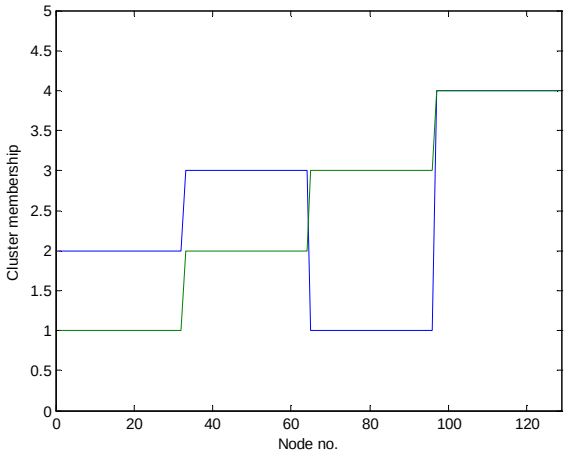

Figure 2.1

The horizontal axis gives the node id. numbers. The demo utilized a *Girvan-Newman* graph [NewmanGirvan2004] with 128 nodes. These nodes originally were partitioned (by their edge patterns) into four *communities* (i.e., clusters) as follows: nodes 1, 2, …, 32 go into community no.1, nodes 33, 34, …, 64 go into community no.2 and so on; this partition is V_0 and is plotted by the green line in Fig.1. V_{est}, on the other hand, assigns nodes 1, 2, …, 32 to community no.2, nodes 33, 34, …, 64 to community no.3 and so on. While the community *labels* are different between V_0 and V_{est}, the actual division into clusters is the same. In short, the modularity maximization algorithm obtained the correct solution, modulo a relabelling of clusters.

2.3 The GUI

Let us also run a simple demo of the Graphical User Interface (GUI). In the MATLAB command line type

```
>> gui
```

and hit `[Enter]`. It is important to note that the `gui` command, should always be executed from the root directory of the CDTB (i.e. `C:\CDTB`). After executing the command, you will get the following standard MATLAB GUI.

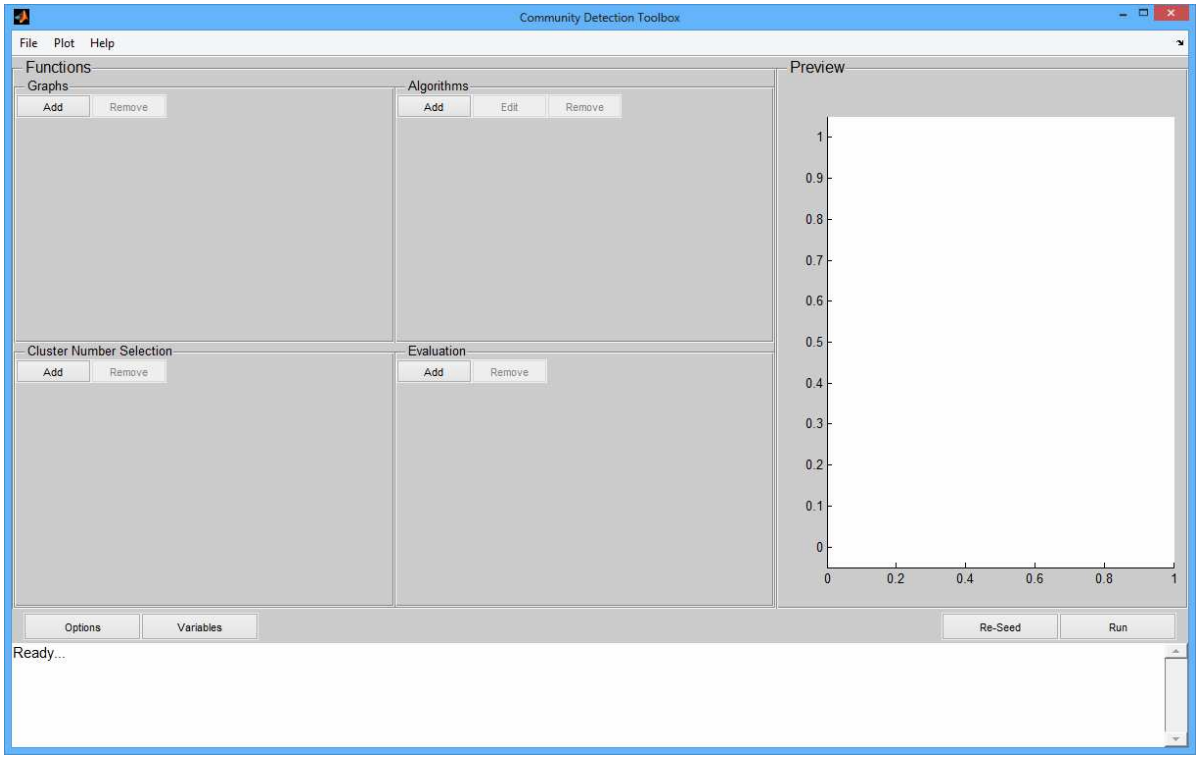

Figure 2.2

The GUI, makes it easy to design and perform community detection experiments. To perform exactly the same experiment that `CDTBDemo01` performed, do the following. Go to the GUI menu `File/Import Data`. A standard *File Dialog* will open, like this

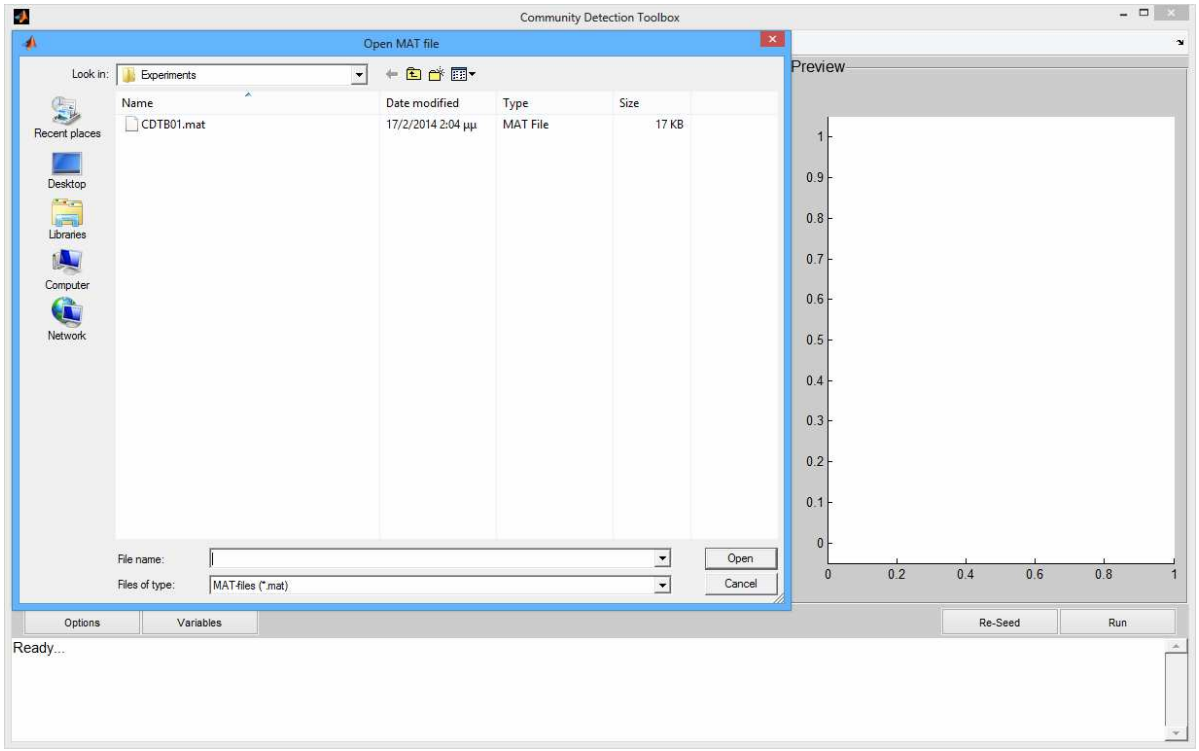

Figure 2.3

Select the file `CDTB01` and click on the `Open` button. The GUI will change and now look like this:

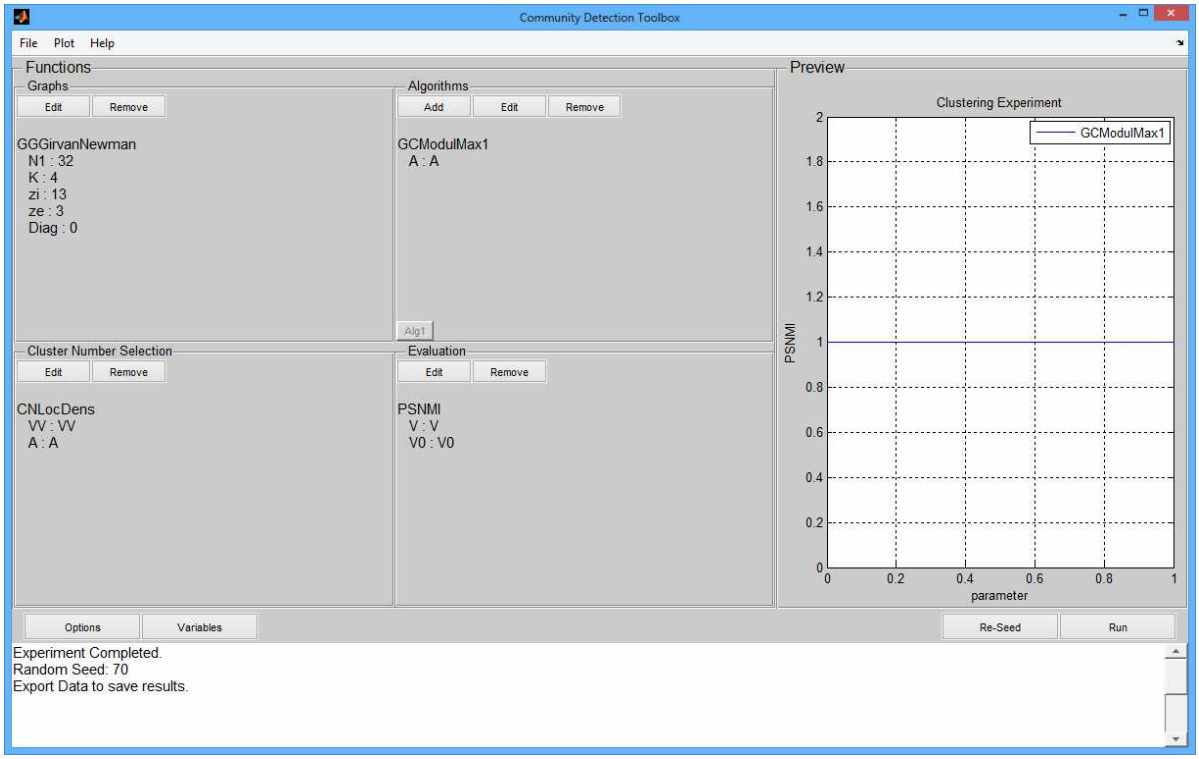

Figure 2.4

Now click on the `Run` button. A *progress bar* will flash for a while and then you will get a plot (actually a straight line). The GUI has run the experiment specified by the choices appearing in Fig. 2.4. You can access these results by selecting the menu option `Plot/Results to Command Line`: if you switch to the MATLAB command line (it is always available, e.g. by `[Alt]-[Tab]`-ing) you will see that a new variable has been loaded in your *MATLAB workspace*, called `Results`. We will later discuss which results are contained in `Results`. We can now take a break.

3. Basic Theory

In this manual we will use the following mathematical notation.

1) As already mentioned, *graph* is used as a synonym of *network*; *cluster* as a synonym of *community*; *clustering* as a synonym of *community detection*.
2) A *graph* $G=(V,E)$ consists of a *node set* V and an *edge set* E. The nodes contained in V will always be labeled as 1, 2, 3, ..., N. The edges contained in E indicate which nodes are connected; so if (x,y) is in E, then we know that node x is connected to node y. In CDTB we only deal with *undirected* graphs, so (x,y) is the same as (y,x); they both tell us that x and y are connected.
3) Every graph G has an *adjacency matrix* A. If the graph has N nodes, A is an $N\times N$ matrix of 0's and 1's. $A_{xy}=1$ if and only if $(x,y)\in E$, i.e., x and y are connected. Since $(x,y)=(y,x)$, we see that A is a *symmetric* matrix.
4) A *clustering* of $G=(V,E)$ is a *partition* of V into sets $V_1,...,V_K$ such that $V_1\cup...\cup V_K=V$, $V_1\cap...\cap V_K=$ and none of V_1, ..., V_K is empty. The sets $V_1,...,V_K$ are the *clusters*. We write the partition as $\mathbf{V}=\{V_1,...,V_K\}$. The *size* of the partition is $K=|\mathbf{V}|$.
5) Given a graph $G=(V,E)$ and a *partition* $\mathbf{V}=\{V_1,...,V_K\}$, the edges of G can be partitioned into sets E_{ij} as follows

$$(x,y)\in E_{kl} \text{ if and only if } x\in V_k \text{ and } y\in V_l$$

6) In particular, we write $E_k^i=E_{kk}$ and $E_k^e=\cup_{l\neq k}E_{kl}$. In other words, the set E_k^i contains the *internal* edges of V_k, with both their ends belonging to the same cluster, while the set E_k^e contains the *external* edges of V_k, which have one end in V_k and the other end in $V-V_k$, the set of nodes which do not belong to V_k.

By *community detection* we mean the activity of graph clustering, i.e., of finding a partition $\mathbf{V}=\{V_1,...,V_K\}$ of a graph $G=(V,E)$ into clusters; the nodes contained in each cluster must somehow be more related to each other than to nodes outside the cluster, thus forming a community. While much has been written about graph (or network) communities, no clear and generally accepted definition of what constitutes a community is available. Most researchers agree that a community is characterized by dense connectivity between its members and sparser connectivity with nodes outside the community [Fortunato2010]. Beyond this (somewhat vague) definition communities can be defined in terms of *quality functions*. A quality function is a function $Q(G,\mathbf{V})$ (i.e., it depends on both the graph G and the partition $\mathbf{V}$) the value of which characterizes how good $\mathbf{V}$ is as a partition of G. Hence the best decomposition of G into communities is the partition $\mathbf{V}^*=\{V_1^*,...,V_K^*\}$ which maximizes Q, i.e., $\mathbf{V}^*=\arg\max_{\mathbf{V}} Q(G,\mathbf{V})$. And then good communities are the elements of a good $\mathbf{V}$, i.e., a $\mathbf{V}$ which achieves a high $Q(G,\mathbf{V})$ score. Obviously this

definition of communities depends on the particular quality function Q used. A large number of quality functions have been proposed (which is a good indication that none of them is entirely satisfactory). The most popular quality function is the Girvan-Newman modularity

$$Q_{GN}(G,\mathbf{V}) = \sum_{k=1}^{K}\left(\frac{E_k^i}{2m} - \left(\frac{m_k}{2m}\right)^2\right)$$

where $m_k = \sum_{x\in V_k}\sum_{y\in V} A_{xy}$ is the *total degree* of cluster V_k . Another reasonable quality function is the *local density*

$$Q_{LD} = \frac{1}{2}\sum_{k=1}^{K}\left[\frac{\sum_{i\in V_k}\sum_{j\in V_k} A_{ij}}{{N_k}^2} + 1 - \frac{\sum_{i\in V_k}\sum_{j\notin V_k} A_{ij}}{N_k\cdot(N-N_k)}\right],$$

as will be explained in Appendix A. Having chosen a quality function $Q(G,\mathbf{V})$, community detection is equivalent to the maximization of $Q(G,\mathbf{V})$ with respect to $\mathbf{V}$ (for a particular G). This is a combinatorial problem and becomes increasingly hard as the size of the graph increases (in terms of either the number of nodes or the number of edges). Hence a large part of community detection research consists in the development of tractable, *approximately* optimal algorithms for quality function maximization. CDTB contains a large number of such algorithms and if the user invents his own quality function and / or maximization algorithm he can very easily incorporate these in CDTB.

4. CDTB and MATLAB Programming

In this section we discuss the use of CDTB in writing MATLAB code. We start our presentation with an example. Recall that in Section 1.2 you typed the command `CDTBDemo01` which resulted in MATLAB performing and plotting a graph clustering. Here is the listing of the *MATLAB script* `CDTBDemo01.m` (it can be found in the folder `C:\CDTB`):

```
01 clear all
02 clc
03 [A,V0]=GGGirvanNewman(32,4,13,3,0);
04 V=GCModulMax1(A);
05 N=length(V);
06 K=max(V);
07 Q1=PSNMI(V,V0);
08 disp(['The NMI metric between V0 and Vest is ' num2str(Q1)]);
09 figure(1); plot([V V0])
10 axis([0 N+1 0 K+1])
11 xlabel('Node no.')
12 ylabel('Cluster membership')
```

The important lines in the above listing are `03`, `04` and `07` and each of them performs a different task. Namely:

1. `[A,V0]=GGGirvanNewman(32,4,13,3,0)` **generates** a graph (hence the first two letters of the function are **GG**). It is a Girvan-Newman graph; the arguments of the function are various parameters of the graph and the output is `[A,V0]` where `A` is the graph adjacency matrix and `V0` is the *generating partition*.
2. `V=GCModulMax1(A)` performs **graph clustering** (hence the first two letters of the function are **GC**). Clustering is performed by modularity maximization. The graph information is supplied to the function through its adjacency matrix `A` previously generated by `GGGirvanNewman`. The output of the function is the modularity optimizing partition `V`.
3. `Q1=PSNMI(V,V0)` evaluates the obtained clustering by computing the **partition similarity** (hence the first two letters of the function are **PS**). The particular partition similarity metric used is the

Normalized Mutual Information between the generating partition `V0` (generated by `GGGirvanNewman`) and the optimizing partition `V` (generated by `GCModulMax1`).

The three functions mentioned above are CDTB functions, specified in corresponding `*.m` files. They are good examples of the types of functions contained in the Toolbox. More specifically, CDTB contains the following subfolders:

1. `Algorithms` which contains graph clustering algorithms;
2. `Evaluation` which contains evaluation functions;
3. `Graphs` which contains graph generators.

In addition to the above, CDTB also contains the following folders.

1. `Auxiliary` contains various auxiliary / utility functions.
2. `Cluster_Number` which contains *cluster number selection* functions (more on this a little later).
3. `Experiments` which contains some demo files.

The details of these functions (and, in particular, their syntax) are presented in Appendix C (Function Reference). In the rest of the current section we will present some general remarks and examples to help the user write his own MATLAB programs, using the CDTB functions. We have tried to follow a consistent style in the naming, input and output of functions. The two initial function names conform to the following conventions.

Initial Letters	**Functionality**
`GG`	Graph Generator
`GC`	Graph Clustering
`CN`	Cluster Number selection
`PS`	Partition Similarity
`QF`	Quality Function

Let us now present some additional remarks regarding each function category.

1. Graph Generators. Different graph generators require different graph parameters as input (these are

documented in Appendix C) but there is one input which must be specified for every graph generator, namely `Diag` which indicates whether the adjacency matrix `A` of the graph will contain zeros or ones in the diagonal (these correspond to the absence or presence of self-loops for the graph nodes). The output of *all* graph generators is always the same: `[A,V0]` where `A` is the graph adjacency matrix and `V0` is the generating partition. When the graph contains *N* nodes, `A` is an *N*x*N* matrix of 0's and 1's (CDTB does not deal with weighted or signed graphs) which contains enough information to fully specify the graph. The generating partition `V0` is an *N*x1 vector with the *n*-th element containing the number of the cluster to which the *n*-th node belongs; `V0` is contained in the output so that subsequent partitions of the graph can be compared to the "true" partition.

2. Graph Clustering. Every graph clustering function requires as minimum input a description of the graph to be clustered. This is given in terms of `A`, the adjacency matrix. Several algorithms require additional input (which is documented in Appendix C). The output of a graph clustering algorithm can take two forms. First, it can be a single clustering described by the *N*x1 vector `V` which has the same for as the previously mentioned `V0`. Alternatively, some algorithms output an *N*x*M* matrix `VV` which contains *M* clusterings, i.e., `VV(:,m)` contains the *m*-th clustering; this situation occurs when the algorithm computes several different clusterings, one for every value of an algorithm parameter. For example, the AFG algorithm is invoked by `VV=GCAFG(A,Scale)`, where `Scale` is a vector containing several values of the *scale parameter*; for every such value the AFG algorithm provides a different clustering.

3. Cluster Number Selection. Functions of this type are used when our clustering algorithm provides more than one possible clustering (e.g., the AFG previously mentioned algorithm). In this case we want a *cluster number selection criterion*. For example, consider the function `Kbst=CNModul(VV,A)`; its first input is the *N*x*K* matrix `VV` which contains *K* clusterings; its second input is `A`, the adjacency matrix describing the graph. These are the two inputs required to compute the Newman-Girvan modularity $Q_{NM}(V^{(k)},G)$ for k=1, 2, .., K ; the function returns as output the integer `kbst` iff $V^{(kbst)}$ is the clustering with highest modularity (with respect to the graph G). All cluster number selection functions have the same input / output structure described above.

4. Partition Similarity. These functions are used to compute the similarity between two clusterings (partitions) V_1 and V_2. Usually (but not necessarily) V_1 will be the true partition and V_2 will be the one returned by the clustering (or cluster number selection) algorithm.

5. Quality function. While partition similarity functions evaluate a partition by comparing it to the true

partition, quality functions evaluate a partition *V* for which the "ground truth" is not known. Hence *V* is evaluated using the information inherent in the graph *G* or, rather, in its equivalent representation through the adjacency matrix *A*.

This concludes our general description of the various CDTB functions. Let us now present an example which utilizes functions from all categories. This example (an extension of the previously considered `CDTBDemo01.m`) is contained in the file `CDTBDemo02.m` which can be found in the `C:\CDTB` folder. It listing is as follows.

```
01 clear all; clc
02 N1=32; K=4; Diag=1;
03 Scale=[2 1.5 0.5 0.4 0.3 0.2];
04 for i=0:8
05   zi=16-i;
06   zo=i;
07   [A,V0]=GGGirvanNewman(N1,K,zi,zo,Diag);
08   N=length(V0);
09   VV=GCAFG(A,Scale);
10   Mbst=CNLocDens(VV,A);
11   V=VV(:,Mbst);
12   Q1(i+1,1)=PSNMI(V,V0);
13   K1(i+1,1)=max(V);
14   figure(1); plot([V V0])
15   axis([0 N+1 0 K1(i+1)+1])
16   xlabel('Node no.'); ylabel('Cluster membership'); pause(0.5);
18 end
19 figure(2); plot(Q1); axis([1 9 -0.05 1.05]);
20 xlabel('zo'); ylabel('NMI(V,V0)')
21 figure(3); plot(K1); axis([1 9 0 max(K1)])
22 xlabel('zo'); ylabel('NMI(V,V0)')
```

The above script implements an experiment of graph clustering which involves a sequence of Girvan-Newman graphs with an increasing number of external edges. We give below comments of the important lines in the script.

`02` Here we give the Girvan-Newman graph parameters.

`03` The `Scale` parameter is used by the AFG algorithm.

`04` The main loop runs on the counter `i`.

`05` The `zi=16-i` is the average number of internal edges per node.

`06` The `zo=i` is the average number of external edges per node.

`07` Here we generate a Girvan-Newman graph with `zi, zo` edges.

`09` Here we perform the actual clustering with the AFG algorithm. Since the `Scale` parameter takes 6 different values, the output matrix `VV` contains 6 different clusterings.

`10` The `CNLocDens` cluster number selection criterion selects the best column `Mbst` of `VV`.

`11` The best clustering `VV(:,Mbst)` is stored in `V.`

`12` The `V` clustering is compared to the generating clustering `V0` via the NMI function.

`18` The main loop is concluded.

`19` Plot the quality of the best clustering for each `z0` value.

`21` Plot the number of clusters in the best clustering for each `z0` value. We see that

In Figure 4.1.a we plot NMI(V_{bst},V_0) and in Figure 4.1.b the number of clusters in V_{bst}, both as function of z_o, the number of outer edges. We see that for high "noise" (high z_o values) V_{bst} has more clusters than the generating partition V_0.

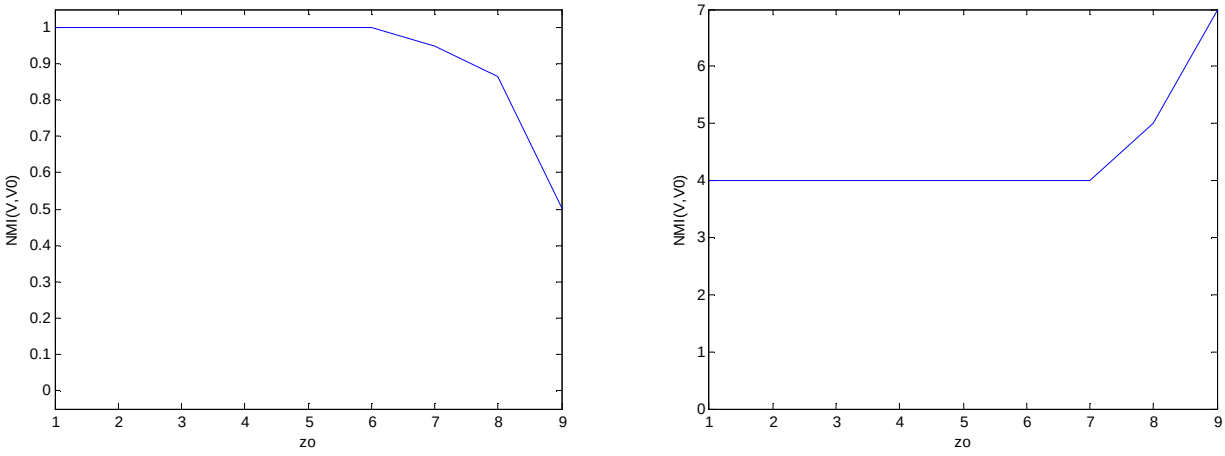

Figure 4.1

The user can inspect the scripts in the `C:\CDTB\Examples` folder to see some additional examples of CDTB programming.

We conclude this section on CDTB programming by discussing the `make` function, contained in the `C:\CDTB\Interface` folder. This function can be used to quickly generate graph clustering experiments. Its syntax is `make(Grf,Alg,Cln,Eval,Opt,Var)` where

- `Grf` is a specification of the graph;
- `Alg` is a specification of the graph clustering algorithm;
- `Cln` is a specification of the cluster number selection criterion;
- `Eval` is a specification of the clustering evaluation function;
- `Opt` is a specification of additional options;
- `Var` is a specification of additional variables.

The above correspond closely to the parameters of the CDTB GUI, which will be discussed in Section 5. For more information about the make function, and a complete example you can refer to the make help, which is accessed by typing

```
>> help make
```

in the MATLAB command line.

5. The GUI

The Community Detection Toolbox provides a GUI to access its capabilities. It has been designed to be user-friendly, but requires some effort to understand its structure and use it efficiently. Its features are illustrated in this section.

5.1 A Short Description of the GUI

In the MATLAB command line type:

```
>> gui
```

The *basic window* pops up. As mentioned in Section 2.3, the `gui` command should always be executed from the root directory of the CDTB.

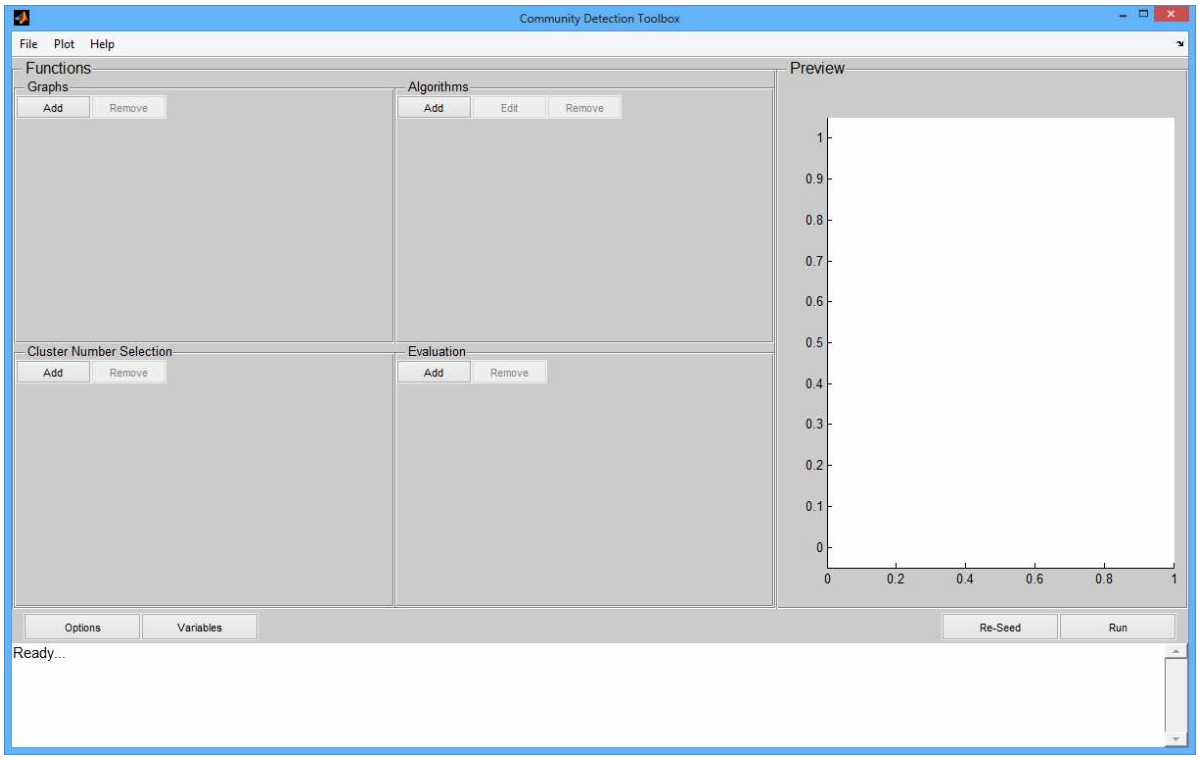

Figure 5.1

The *basic window* is divided in three **panels** (different areas of a MATLAB figure):

1. **Functions**. This panel is used to specify the functions and their parameters for the experiment. It contains the *sub-panels* **Graph**, **Algorithms**, **Cluster Number Selection** and **Evaluation**.
2. **Preview**. When an experiment is run, its outcome is plotted in this panel. Moreover, the figure can be plotted to a separate window from the `File/Export Figure` menu or by pressing Ctrl+F.
3. **Messages**. In this panel appear messages to the user.

In addition to the three panels, the user has access to four **buttons**.

1. **Options**. Used to set several experiment options.
2. **Variables**. Variables menu provides a solution for the user to define and use through the execution of the experiment, several variables. Although, these variables do not change on every iteration, they are a solution to better organize the experiment.
3. **Re-Seed**. This is used to reset the MATLAB random number generator. Many of the functions included in the Toolbox (especially graph generators) are based on pseudo-random functions. Therefore, in order to create the exact same dataset for the experiment, the random seed is needed. Re-Seed changes the seed, thus providing a new dataset.
4. **Run**.

Finally, the user has access to three **menus**.

1. **File**: this menu contains options for loading and saving experiments.
2. **Plot**: this menu provides data visualization options.
3. **Help**: provides links to help-files

5.2 An Illustrative Experiment

We will now present a detailed example of how to set up a community detection experiment using the GUI.

To be specific, suppose that we want to compare two graph clustering algorithms, Danon's **modularity maximization** algorithm and a variant of **spectral clustering**. We will apply these algorithms to a family of *planted partition graphs*; for each graph, each algorithm will produce a partition; we will evaluate these partitions, plot the average results of each algorithm and will use the plot to compare the two algorithms.

To do all of the above we start by typing

```
>> gui
```

at the MATLAB command line and hitting `enter`. The *basic window* pops up.

5.2.1 Determining the Graph Family

Let us specify the graph family we will use. At the `Graphs` sub-panel, click `Add` to select the `Graph Generator` function. A new window pops up.

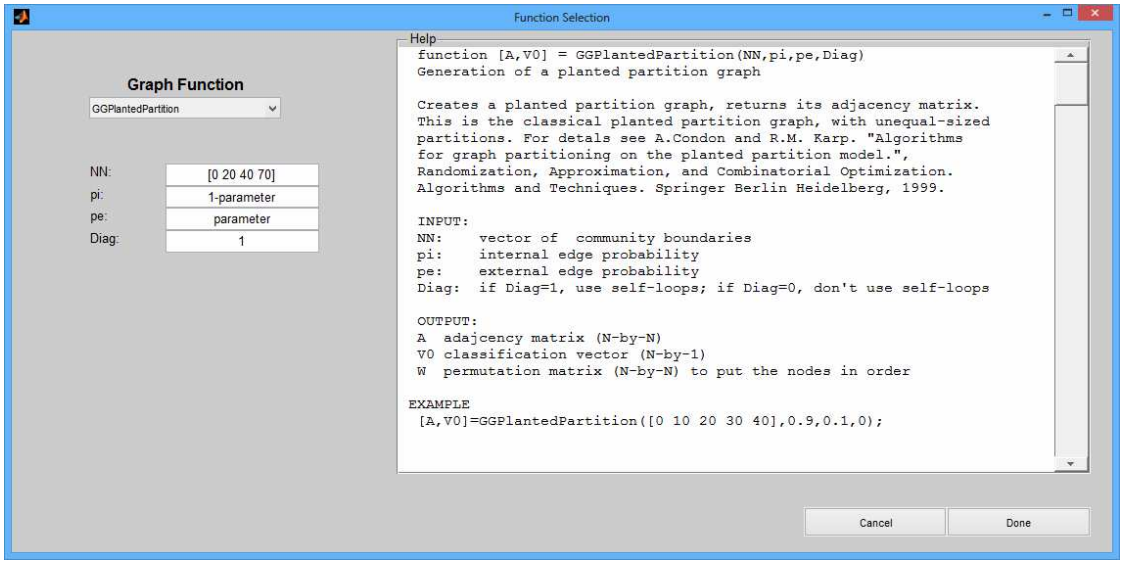

Figure 5.2

From the drop-down list choose the `GGPlantedPartition` graph generator function. This is a standard MATLAB function which requires certain inputs; you can see the required inputs and their interpretations in the help file on the right side of the window. In the textboxes on the left side of the window, type the following values:

`NN`: `[0 20 40 70]`
`pi`: `1-parameter`
`pe`: `parameter`
`Diag`: `1`

and click `Done` to accept the inputs.

Why did we use the expression `parameter` for the **`pe`** value? The reason is that we do not want to use a single **`pe`** value, but a range of values. As will be seen soon, `parameter` is a MATLAB vector which contains several **`pe`** values (and its value will be determined with the `Options` button). The same holds for

pi and the expression `1-parameter`. By using several (**pi, pe**) values can evaluate the efficiency of the community detection algorithms at various levels of community structure.

5.2.2 Choosing Graph Clustering Algorithms

At the `Algorithms` sub-panel, click `Add` to select the `Graph Clustering` function. A new window pops up. Select `GCDanon` from the drop down menu and click `Done`. Note that the input to this algorithm is the adjacency matrix A (generated internally by the `Graph Generator` function at every iteration and passed to the graph clustering Algorithm); ***do not change the corresponding input***.

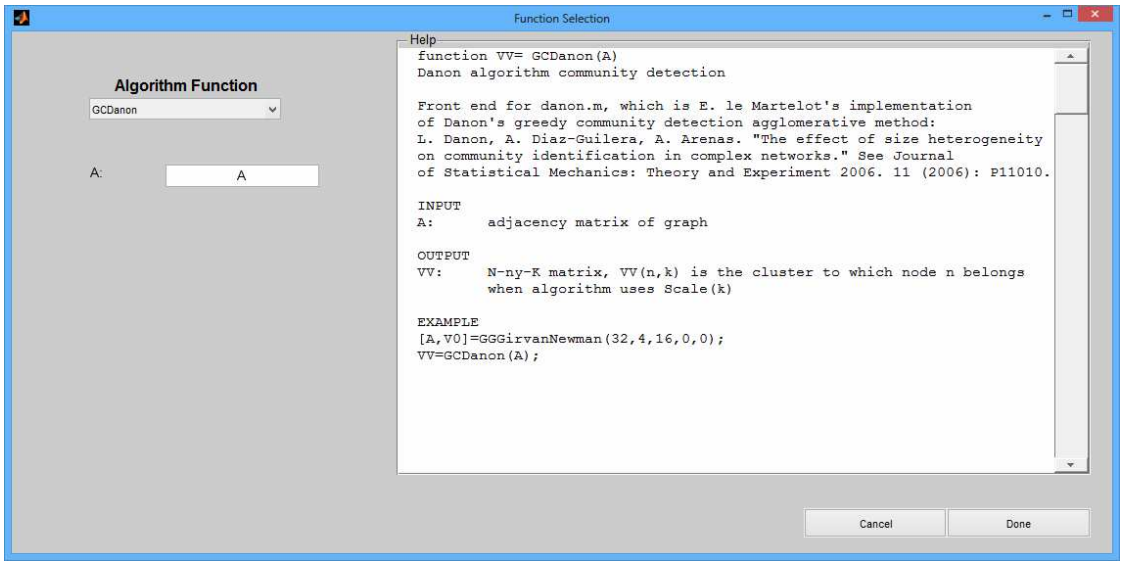

Figure 5.3

Click `Add` once again, and select `GCSpectralClust1`; this algorithm uses the internal variable `A` but also requires the user to provide the maximum number of clusters; in the textbox named `Kmax` type the value 8 and then click `Done`.

5.2.3 Choosing a Cluster Number Selection Criterion

Some graph clustering algorithms automatically determine the optimal number of clusters; this is the case with the Danon algorithm. Other algorithms require an additional *cluster number selection criterion*; this is the case with the spectral clustering algorithm. Hence we now go to the `Cluster Number Selection` sub-panel, click `Add`, select the `CNDistBased` criterion and click `Done` (the inputs to `CNDistBased` are

predetermined as A and VV, and **should not be changed)**.

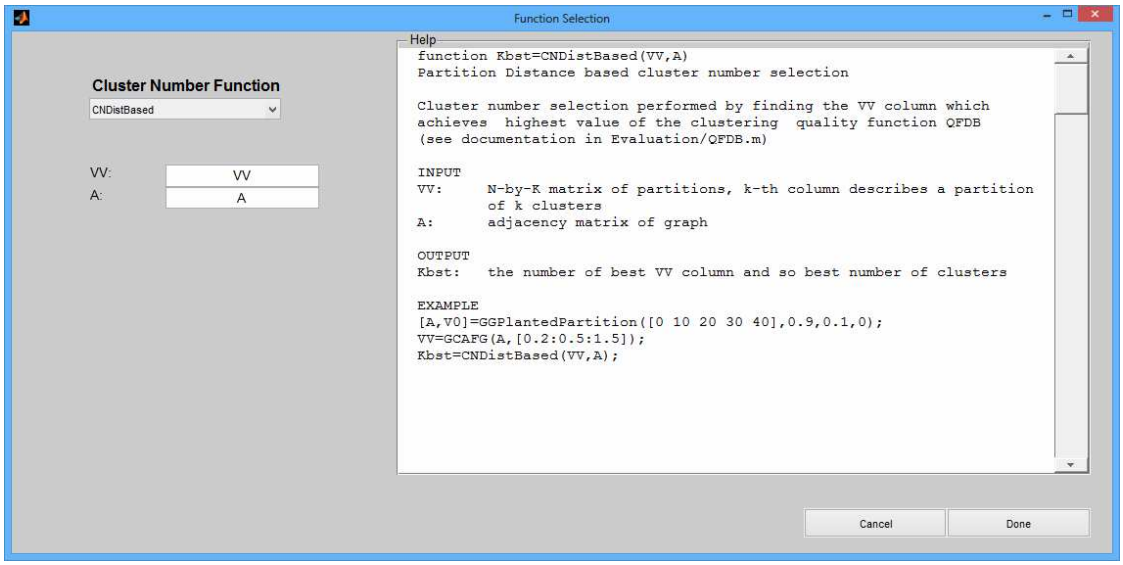

Figure 5.4

5.2.4 Choosing the Evaluation Function

At the `Evaluation` sub-panel, click `Add`, select the `PSJaccard` evaluation function and click `Done`. This function will evaluate the *Jaccard similarity* between `V` (the clustering obtained by the graph clustering algorithm) and `V0` which is the reference clustering, returned from the Graph Generator function.

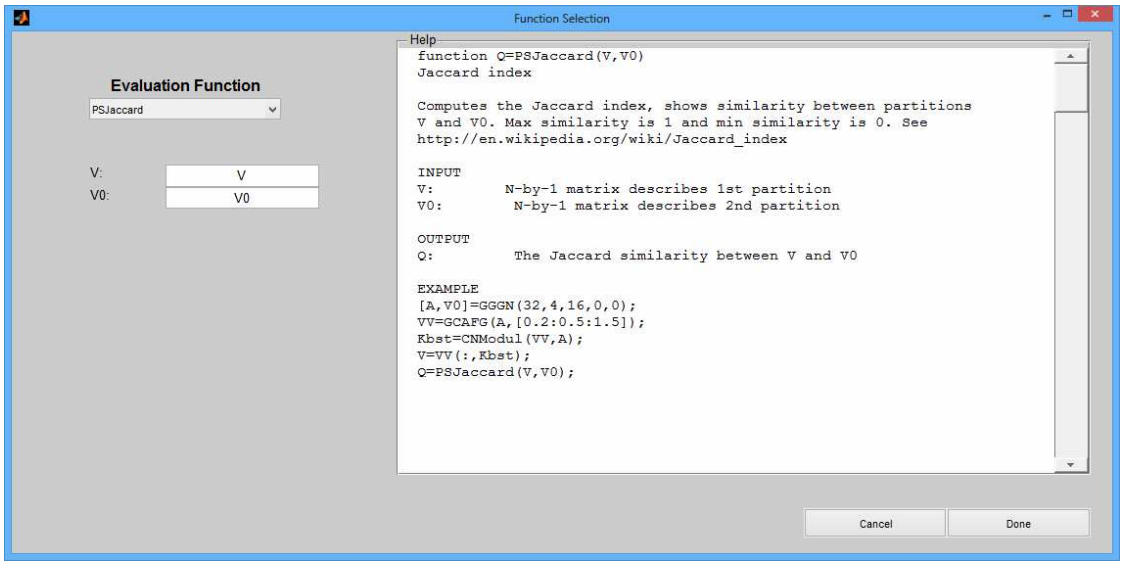

Figure 5.5

5.2.5 Fixing Options of the Experiment

Finally we can set the `parameter` and general options of the experiment. From the GUI's *basic window* click the `Options` button, and fill in the textboxes to get the following picture.

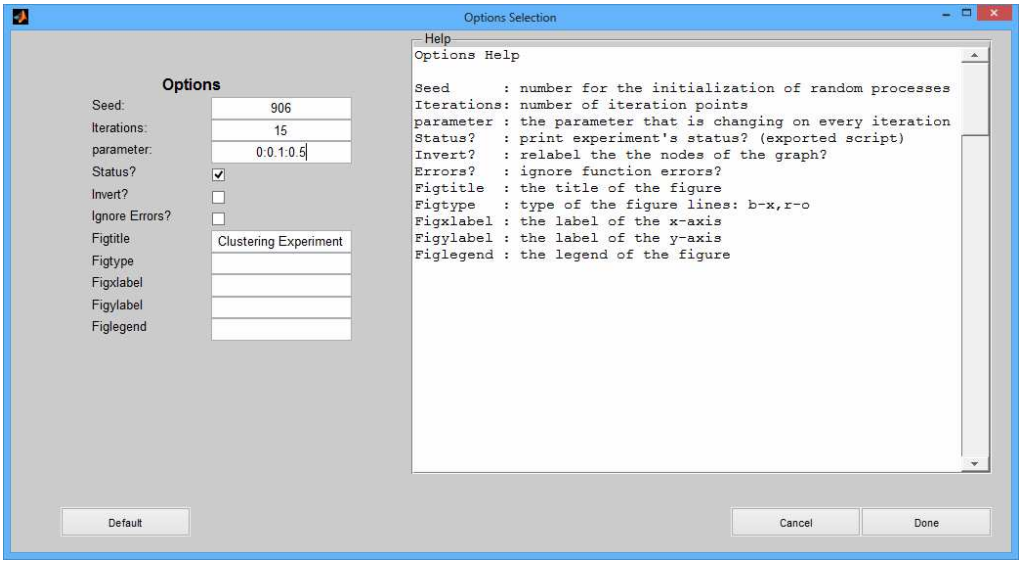

Figure 5.6

Note that we have specified `parameter=0:0.1:0.5`, *in MATLAB* notation. `Iterations` is the number of iterations for each value that `parameter` takes (after all 15 iterations are completed, the value for this point is the mean of the 15 values obtained). Hence, the current experiment will consist of 90 community detection runs for each of the two algorithms (*why?*).

5.2.6 Running the Experiment

We are now ready to run the experiment. From the main GUI Figure click the `Run` button. A waitbar appears.

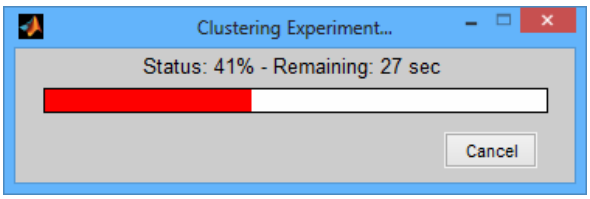

Figure 5.7

After the execution of the experiment has finished, a plot of the evaluation for each clustering with respect to

the parameter appears in the axes of the Preview panel. You can click Menu: `File/Export Figure` or hit `Ctrl+F` to create a new *MATLAB* figure with the plot, that enables you to save or edit it accordingly.

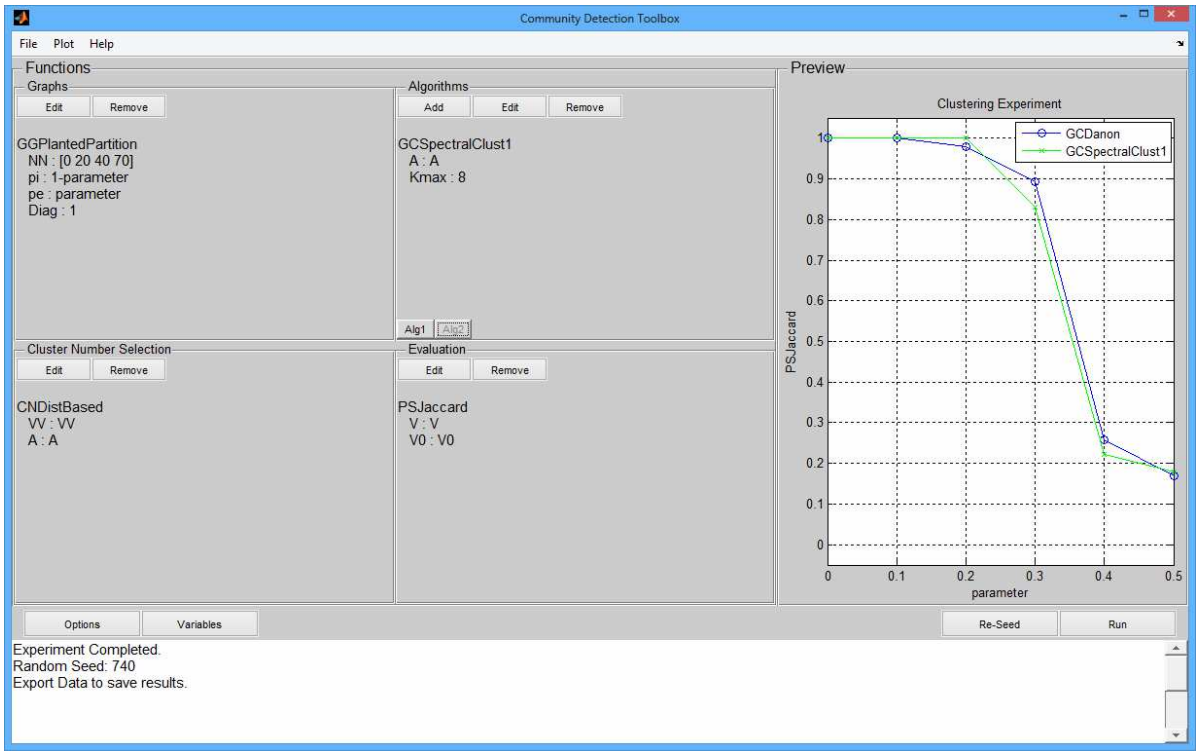

Figure 5.8

5.2.7 Specifying Variables

Another feature of the GUI is that you can use the `Variables` section, in order to organize the experiment. We will define two variables, representing two different Graph vectors (`NN` as defined in the Graph Generator function).

Click the `Variables` button, from the main GUI figure. From the figure that pops up, click the `Add` button twice, in order to create two new variables, and set:

Name	**Value**
NN1:	`[0 20 40 80]`
NN2 :	`[0 30 40 100]`

Finally, click `Done`.

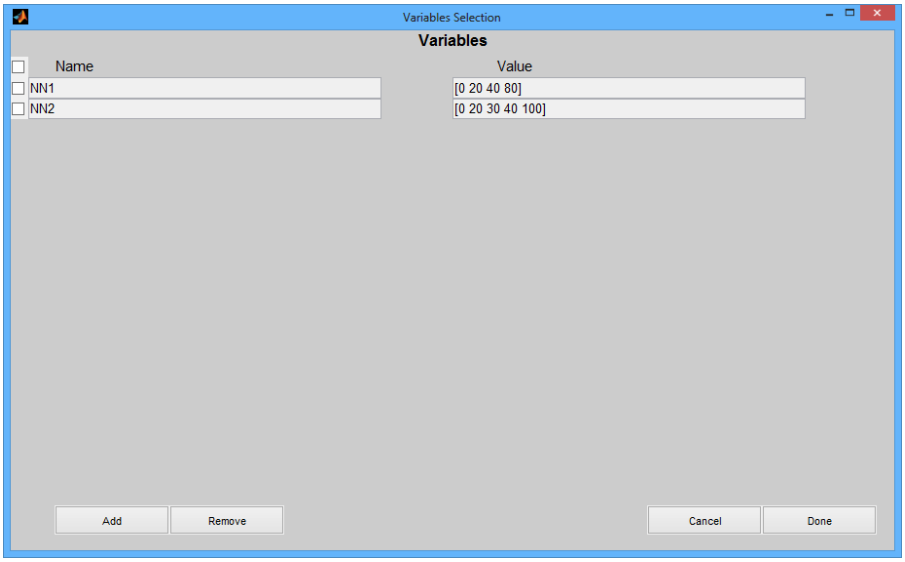

Figure 5.9

We can now edit the Graph Generator function (using the `Edit` button), and change the NN value from: [0 20 40 70] to either: `NN1` or `NN2` and click `Done`.

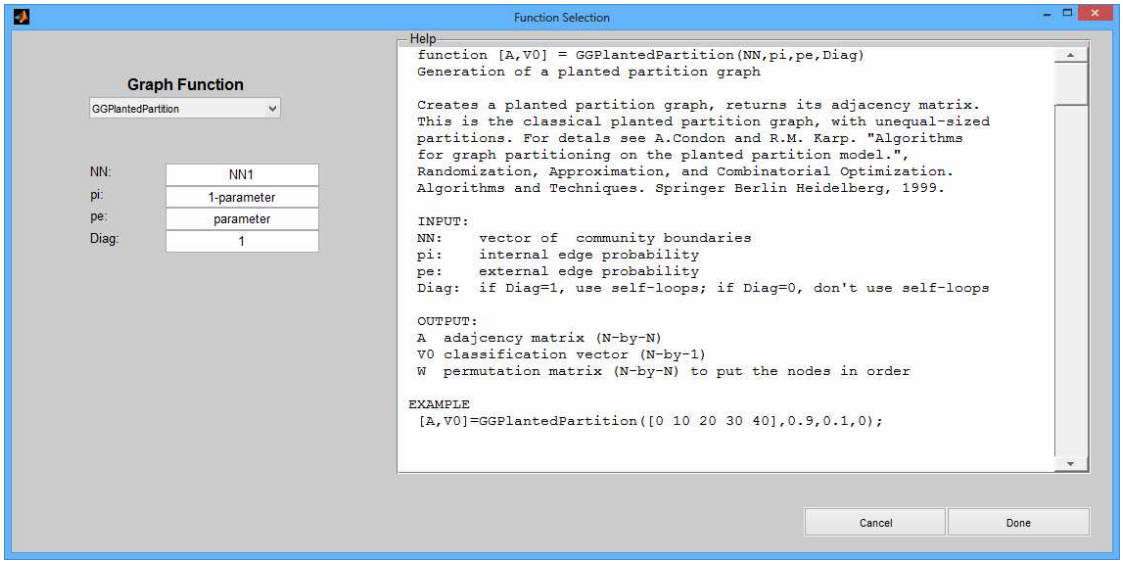

Figure 5.10

5.2.8 Re-Seeding

As noted earlier, functions returning random values are actually pseudo-random functions. As a result, in order to create a dataset, an initial value (the seed) is needed. Re-Seed changes the seed, thus providing a new dataset.

You can optionally select `Re-Seed`, to change the seed for the random functions. The new seed appears in the Messages panel. Click 'Run' to execute the new experiment.

5.2.9 Post-Processing

In addition to the plot that appears in the `Preview` panel, the GUI has other options to evaluate the results of the experiment, as well. These features are described in the current section.

Adjacency Matrices

From the *basic window*, click `Plot/Adjacency Matrices`. A new figure appears, with the adjacency matrix A for each iteration point. In this figure, the point (i,j) is painted red if and only if A(i,j) = 1 and blue, otherwise.

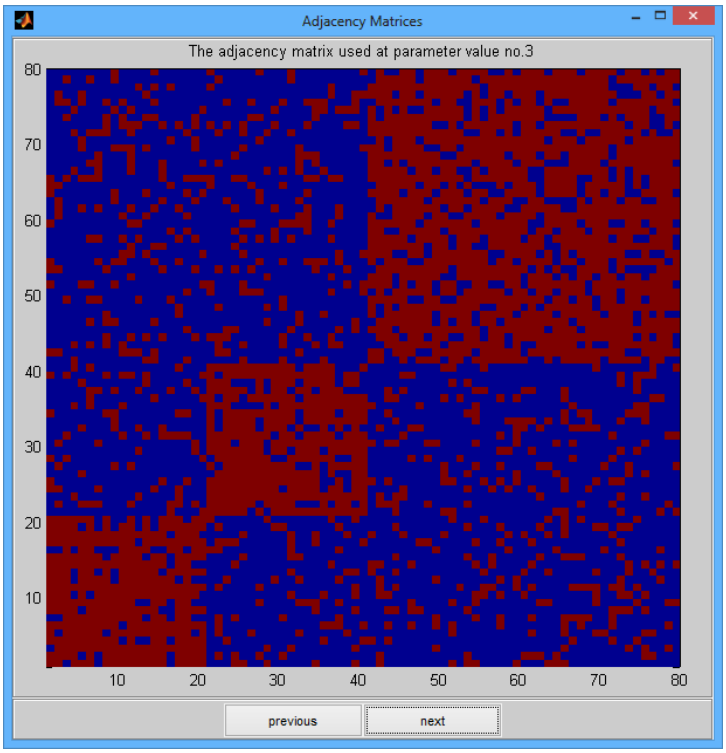

Figure 5.11

Clicking `next` enables you to see the various adjacency matrices for the different parameter values.

Graphs

Apart from the Adjacency Matrix plot, we can get a better intuition about the Graph being clustered, using the `Plot/Graphs` menu. Note, that this utility can be more useful for small graphs.

In the plot that appears, each dot corresponds to a node, and each line to an edge of the graph. The important thing is that each dot is painted according to the cluster that the corresponding node belongs.

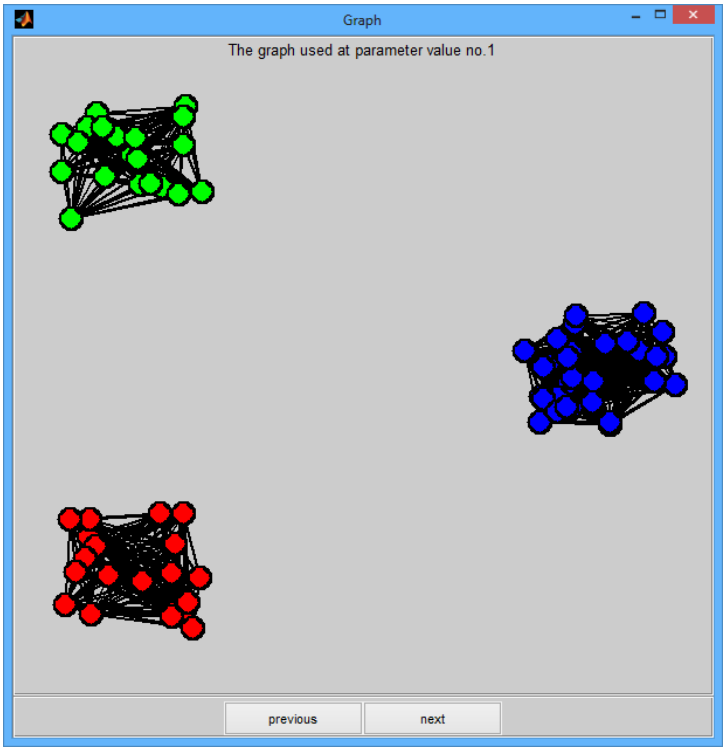

Figure 5.12

Clustering

The other two options (**All Partitions, Best Partition**) create figures of the partition obtained for every iteration. More specifically, `Plot/All Partitions`, shows a XY-plot in which the x-axis values represent the id of each node, and the y-axis values represent the cluster, each node belongs in.

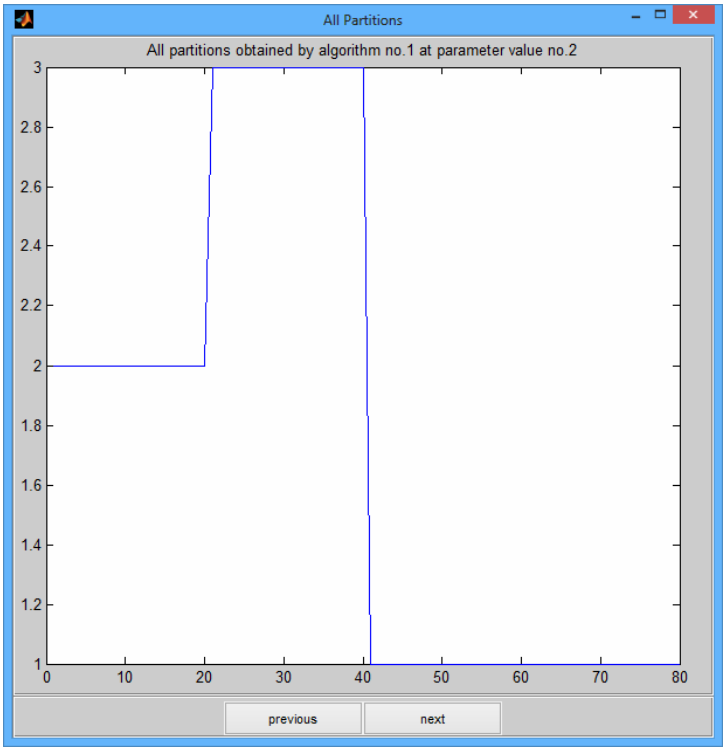

Figure 5.13

On the other hand, `Plot/Best Partition` plots the obtained clustering in the same figure with the reference clustering V0.

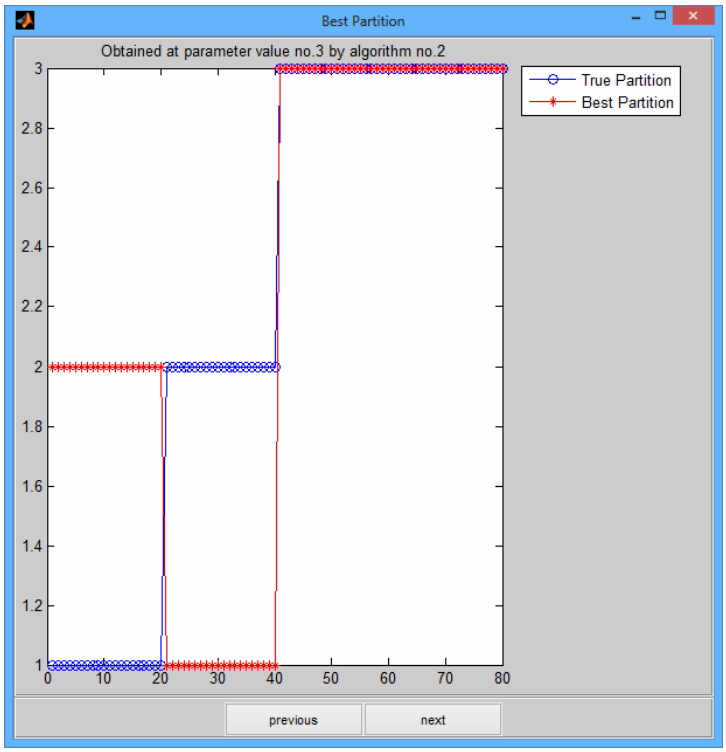

Figure 5.14

Results to Command Line

Finally, you can click `Plot/Results` to Command Line, in order to copy a struct containing:

1. data and
2. results of the experiment

from the GUI, to the MATLAB command line interface. The stuct returned is 6x2 struct array, named `Results` with the following fields:

- A: the adjacency matrices
- V0: the reference ('best' clustering)
- V: the clustering obtained after the cluster number selection
- VV: the various clusterings returned from the algorithm
- CN: the value returned from the cluster number selection function
- a: the value returned from the evaluation function

You can access the struct's data using the following notation:

```
Results(i,j).field
```

where:

- i: is an index in the range of 1 to number of parameter points
- j: is an index from 1 to number of clustering algorithms used
- field: is a field from the ones above.

6. Conclusion

The Toolbox was design with the intent to be parametric. This means that the user may add and/or remove functions to the sub-folders without having to make other changes. As a result, one can easily use new algorithms with the Toolbox. However, it should be noted that the functions should satisfy the following requirements (otherwise, the GUI will not be able to read the file):

1. the file must have an .m extension;
2. the file must start with a line in the format "`function out = fun(arg1,arg2,...)`".

Some compatibility issues have been noticed when trying to import experiment data after changes have been made to the Toolbox. The developers have tried to address the problem in the best possible way, however, one needs to be aware.

A. Some More Theory

Most of the functions included in CDTB are documented in published papers; when this is the case we give the relevant references in the Function Reference (Appendix C). However, CDTB includes some functions which have not appeared in a generally available publication; this section gives some elements of the motivation behind these functions.

A.1 Quality Functions

We propose the use of several quality functions which have not received much attention in the literature. Namely, we introduce the following quality functions.

Global density quality function. This function attempts to quantify the commonly held opinion that "*communities within networks can loosely be defined as subsets of nodes which are more densely linked, when compared to the rest of the network*" [Danon2005]. We first define the *global internal density* and *global external* density as follows.

$$Q^i_{GD}(G,\mathbf{V}) = \frac{\sum_{k=1}^{K}\sum_{x\in V_k}\sum_{y\in V_k} A_{xy}}{\sum_{k=1}^{K}|V_k|^2}, \qquad Q^e_{GD}(G,\mathbf{V}) = \frac{\sum_{k=1}^{K}\sum_{x\in V_k}\sum_{y\in V-V_k} A_{xy}}{\sum_{k=1}^{K}|V_k|\cdot|V-V_k|}.$$

Consider $Q^i_{GD}(G,\mathbf{V})$: it sums (over all clusters V_k) the inner edges (i.e., $\{x,y\}$ such that both x and y belong to the same V_k) and divides the sum over the total "area" of all clusters (or, alternatively, over the number of all *possible* internal edges). Two points must be emphasized about this definition of $Q^i_{GD}(G,\mathbf{V})$. First, it is assumed that $A_{xx}=1$ for all $x\in V$ (i.e., that each node is equipped with a self-loop). Second all edges except self-loops are counted twice. Keeping these two things in mind, it is easy to see that $Q^i_{GD}(G,\mathbf{V})$ always takes values in the interval $[0,1]$ with $Q^i_{GD}(G,\mathbf{V})=1$ only when G is the union of mutually disjoint cliques; these are (reasonably enough) the graphs of *perfect community structure*. Similar remarks can be made $Q^e_{GD}(G,\mathbf{V})$ which also takes values in $[0,1]$ (and the value 0 is achieved for graph of perfect community structure). Now we define the *global weighted density* $Q_{GD}(G,\mathbf{V})$ by

$$Q_{GD}(G,\mathbf{V}) = \frac{1}{2}[Q^i_{GD}(G,\mathbf{V}) + 1 - Q^e_{GD}(G,\mathbf{V})].$$

In other words, $Q_{GD}(G,\mathbf{V})$ takes the maximum value of one only for graphs of perfect community structure and, more generally, takes values close to one for graphs which have densely connected node subsets. Hence, in accordance to the previously cited statement, $Q_{GD}(G,\mathbf{V})$ is a reasonable community function.

Local weighted density quality function. Other functions can be used to formalize the idea of densely connected node subsets. The *local weighted density* quality function is defined by

$$Q_{LD}(G,\mathbf{V}) = \sum_{k=1}^{K} \frac{|V_k|}{2|V|} \cdot [q^i(V_k,G) + 1 - q^e(V_k,G)]$$

where the *local inner* and *outer densities* are defined by

$$q^i(V_k,G) = \frac{\sum_{x\in V_k}\sum_{y\in V_k} A_{xy}}{|V_k|^2}, \qquad q^e(V_k,G) = \frac{\sum_{x\in V_k}\sum_{y\in V-V_k} A_{xy}}{|V_k| \cdot |V-V_k|}.$$

While Q_{LD} is defined slightly differently from Q_{GD}, they share the same basic motivation. In addition to the previously made points about Q_{GD}, an additional point to be emphasized for Q_{LD} is that the term corresponding to each local cluster (local inner density plus local outer *anti*density) is weighted by the term $\frac{|V_k|}{2|V|}$; this is done because otherwise small dense clusters would influence the total clustering quality disproportionately to their size.

Distance based quality function. This is defined by

$$Q_{DB}(G,\mathbf{V}) = \frac{1}{|V|^2}\|A_G - A_{\mathbf{V}}\|$$

where

$\|B\| = \sum_{x\in V}\sum_{y\in V}|B_{xy}|$ is a matrix norm;

A_G : is the adjacency matrix of G

$$A_{\mathbf{V}} = \begin{cases} 1 & \text{if } x, y \text{ belong to the same cluster (under } \mathbf{V}), \\ 0 & \text{if } x, y \text{ belong to different clusters (under } \mathbf{V}). \end{cases}$$

The motivation behind this definition is the following. Suppose $\mathbf{V}$ is the "absolutely true" clustering of G; let us interpret this to mean that (i) when x and y belong to the same cluster they are connected by an edge and (ii) when x and y belong to different clusters they are *not* connected by an edge. This condition will be fully satisfied only by graphs of perfect community structure, in which case the scaling by $\frac{1}{|V|^2}$ ensures that $Q_{DB}(G,\mathbf{V}) = 0$. When the condition is violated, $Q_{DB}(G,\mathbf{V})$ takes larger values but always in the interval $[0,1]$.

Node membership quality function. This is defined by

$$Q_{NM}(G,\mathbf{V}) = \frac{1}{2|V|}\sum_{x\in V}[\mu(x,V[x]) + 1 - \mu(x,V-V[x])]$$

where $V[x]$ indicates the cluster to which x belongs (i.e., $x \in V[x]$) and, for every $x \in V$ and $A \subseteq V$ we define the node membership by

$$\mu(x,U) = \frac{1}{|U|} \sum_{y \in U} A_{xy}.$$

Hence $\mu(x,U) = 1$ iff x is connected to every $y \in U$ and $\mu(x,U) = 0$ iff x is connected to no $y \in A$; for intermediate situations we get $\mu(x,U) \in (0,1)$. Consequently, $Q_{NM}(G, \mathbf{V}) = 1$ iff *every* node $x \in V$ has $\mu(x, V[x]) = 1$ (total membership to its own cluster) and $\mu(x, V - V[x]) = 0$ (no membership to the rest of the graph); (once again) this is the case only for graphs of perfect community structure.

The quality functions Q_{GD} , Q_{LD} , Q_{DB} and Q_{NM} (implemented by the CDTB functions **`Q=QFGloDens(V,A)`**, **`Q=QFLocDens(V,A)`**, **`Q=QFDistBased(V,A)`**, **`Q=QFNodeMemb(V,A)`** respectively) can be used to evaluate the goodness of some clustering $\mathbf{V}$ of a given graph G (with adjacency matrix A) or to compare two clusterings, $\mathbf{V}'$ and $\mathbf{V}''$. They can also be used to *compute* clusterings, as will be explained below.

A.2 Graph Clustering Algorithms

The CDTB provides four functions which perform clustering by optimizing the four quality functions presented above. For example, the CDTB function **`VV=GCGloDens(A,Kmax,Iter)`** outputs an $N \times K_{\max}$ matrix in which the K -th column contains a K -clustering of the graph G (with adjacency matrix A); each such clustering is obtained by a greedy optimization of Q_{GD} . Similarly, there exists functions **`GCQLocDens`**, **`GCQDistBased`** and **`GCQNodMemb`** .

We include these functions in CDTB for experimental purposes. There is an additional issue that must be resolved: the nature of Q_{GD} , Q_{LD} , Q_{DB} and Q_{NM} is such that usually adding more clusters results in better values. For example, define $F_{GD}(K) = \max_{\mathbf{V}} Q_{GD}(G, \mathbf{V})$; then, as we have experimentally found, $F_{GD}(K)$ is an increasing function of K . This necessitates the introduction of *cluster number selection criterion* which will filter clusterings with too many clusters. CDTB provides such a function, as will next be discussed.

A.3 Cluster Number Selection Criterion

Our approach to cluster number selection is based on the idea of *cluster validity*. In other words, we have

observed that in some cases optimizing a quality function Q yields a clustering which yields high Q values but fails to satisfy some basic criteria which *every* clustering should satisfy, independently of its Q value.

To be specific, consider the following two cluster validity criteria introduced in [Radicchi2004]:

(A.1)

(WRC) Weak Radicchi Criterion: $\forall V_k \in \mathbf{V} \ : \ \sum_{x\in V_k}\sum_{y\in V_k} A_{xy} \geq \sum_{x\in V_k}\sum_{y\in V-V_k} A_{xy}$,

(SRC) Strong Radicchi Criterion: $\forall V_k \in \mathbf{V}, \forall x \in V_k \ : \ \sum_{y\in V_k} A_{xy} \geq \sum_{y\in V-V_k} A_{xy}$.

Both of these criteria try to capture the idea that there should be more edges inside clusters rather than outside.

Now, **WRC** is obviously *too* weak and non-discriminating with respect to individual nodes. In other words, if we have a cluster V_k with high internal edge count (hence satisfying **WRC** with a large margin) then we can add *any* extra node u to V_k and still preserve **WRC**, even when u is totally disconnected from V_k.

On the other hand, we have found **SRC** to be *too strong* and hence leading to the invalidation of many reasonable clusterings.

The problem with **SRC** is that it requires of every node to have more internal than external connections; but if a graph has a very large number of nodes, then any particular node may have very weak connection to any external cluster, but still collect a large total number of external edges simply because there exist too many clusters.

So we propose and use a new criterion, which is somewhere between **WRC** and **SRC**. Namely we introduce:

(A.2) **(NMC)** Node Membership Criterion: $\forall V_k \in \mathbf{V}, \forall x \in V_k \ : \ \mu\left(x, V\left[x\right]\right) \geq \mu\left(x, V - V\left[x\right]\right)$

which uses the previously defined node membership function $\mu\left(x,U\right) = \frac{1}{\left|U\right|}\sum_{y\in U} A_{xy}$. Hence **NMC** is equivalent to

(A.3) $\forall V_k \in \mathbf{V}, \forall x \in V_k \ : \ \frac{1}{\left|V\left[x\right]\right|}\sum_{y\in V[x]} A_{xy} \geq \frac{1}{\left|V - V\left[x\right]\right|}\sum_{y\in V-V[x]} A_{xy}$.

Our criterion is similar to the strong Radicchi criterion but somewhat weaker, because of the normalizing factors $\frac{1}{|V[x]|}$ and $\frac{1}{|V-V[x]|}$. On the other hand, our criterion is stronger than the weak criterion. We have found that it works well in practice, as implemented by the CDTB function **`Kbst=CNNodMemb(VV,A)`**. This function takes as input an $N \times K_{\max}$ matrix in which the K-th column contains a K-clustering of the graph

G (with adjacency matrix A); it outputs the integer K_{bst} which is the largest K value such that the K -th column of **VV** is a valid clustering (i.e., it satisfies (**A.3**)).

B. Bibliography

1. Arenas, Alex, Alberto Fernandez, and Sergio Gomez. "Analysis of the structure of complex networks at different resolution levels." New Journal of Physics 10.5 (2008): 053039.
2. Blondel, Vincent D., et al. "Fast unfolding of communities in large networks." Journal of Statistical Mechanics: Theory and Experiment 2008.10 (2008): P10008.
3. Condon, Anne, and Richard M. Karp. "Algorithms for graph partitioning on the planted partition model." Random Structures and Algorithms 18.2 (2001): 116-140.
4. Danon, Leon, et al. "Comparing community structure identification." Journal of Statistical Mechanics: Theory and Experiment 2005.09 (2005): P09008.
5. Danon, Leon, Albert Díaz-Guilera, and Alex Arenas. "The effect of size heterogeneity on community identification in complex networks." Journal of Statistical Mechanics: Theory and Experiment 2006.11 (2006): P11010.
6. Fortunato, Santo. "Community detection in graphs." Physics Reports 486.3 (2010): 75-174.
7. Fortunato, Santo, and Marc Barthelemy. "Resolution limit in community detection." Proceedings of the National Academy of Sciences 104.1 (2007): 36-41.
8. Hespanha, Joao P. An efficient MATLAB algorithm for graph partitioning. Technical Report, Department of Electrical & Computer Engineering, University of California, USA, 2004.
9. Huang, Jianbin, et al. "Towards online multiresolution community detection in large-scale networks." PloS one 6.8 (2011): e23829.
10. Lancichinetti, Andrea, Santo Fortunato, and János Kertész. "Detecting the overlapping and hierarchical community structure in complex networks." New Journal of Physics 11.3 (2009): 033015.
11. Le Martelot, Erwan, and Chris Hankin. "Multi-scale Community Detection using Stability as Optimisation Criterion in a Greedy Algorithm." KDIR. 2011.
12. Newman, Mark EJ, and Michelle Girvan. "Finding and evaluating community structure in networks." Physical review E 69.2 (2004): 026113.
13. Newman, Mark EJ. "Fast algorithm for detecting community structure in networks." Physical review E 69.6 (2004): 066133.
14. Radicchi, Filippo, et al. "Defining and identifying communities in networks." Proceedings of the National Academy of Sciences of the United States of America 101.9 (2004): 2658-2663.
15. Rand, William M. "Objective criteria for the evaluation of clustering methods." Journal of the American Statistical association 66.336 (1971): 846-850.
16. Reichardt, Jörg, and Stefan Bornholdt. "Statistical mechanics of community detection." Physical

Review E 74.1 (2006): 016110.

17. Ronhovde, Peter, and Zohar Nussinov. "Local resolution-limit-free Potts model for community detection." Physical Review E 81.4 (2010): 046114.
18. Shi, Jianbo, and Jitendra Malik. "Normalized cuts and image segmentation." Pattern Analysis and Machine Intelligence, IEEE Transactions on 22.8 (2000): 888-905.

C. Function Reference

In this section, the help-files for the Community Detection Toolbox algorithms are provided. They are organized in four sections: **Graphs, Algorithms, Cluster Number Selection, Evaluation**.

C.1 Graphs

```
% function [A,V0]=GGGirvanNewman(N1,K,zi,ze,Diag)
% Generation of a Girvan Newman graph
%
% Creates a classical Girvan-Newman graph; returns its adjacency matrix A
% and true community membership vector V0. For details see
% M.E Newman and M. Girvan. "Finding and evaluating community
% structure in networks." Physical review E 69.2 (2004): 026113.
%
% INPUT:
% N1 number of nodes in each community
% K  number of communities
% zi number of internal half-edges per node
% ze number of external half-edges per node
% Diag:  if Diag=1, use self-loops; if Diag=0, don't use self-loops
%
% OUTPUT:
% A  adajcency matrix (N-by-N)
% V0 classification vector (N-by-1)
% W  permutation matrix (N-by-N) to put the nodes in order
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
%
```

```
% function [A,V0] = GGPlantedPartition(NN,pi,pe,Diag)
% Generation of a planted partition graph
%
% Creates a planted partition graph, returns its adjacency matrix.
% This is the classical planted partition graph, with unequal-sized
% partitions. For detals see A.Condon and R.M. Karp. "Algorithms
% for graph partitioning on the planted partition model.",
% Randomization, Approximation, and Combinatorial Optimization.
% Algorithms and Techniques. Springer Berlin Heidelberg, 1999.
%
% INPUT:
% NN:    vector of  community boundaries
% pi:    internal edge probability
% pe:    external edge probability
% Diag:  if Diag=1, use self-loops; if Diag=0, don't use self-loops
%
% OUTPUT:
% A  adajcency matrix (N-by-N)
% V0 classification vector (N-by-1)
% W  permutation matrix (N-by-N) to put the nodes in order
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
%
```

```
% function [A,V0] = GGReadEdgeList(EdgeFile,PartitionFile,Diag)
% Graph Generation from an edge list
%
% Reads a graph from edge list file and a partition from partition
% membership files
%
% INPUT:
% EdgeFile         filename (string) contains an M-by-2 list
%                  of edges (as node pairs)
% PartitionFile    filename (string) contains an N-by-1 list
%                  of cluster number to which each node belongs
% Diag:            if Diag=1/0, use/don't use self-loops
%
% OUTPUT:
% A                adajcency matrix (N-by-N)
% V0               classification vector (N-by-1)
%
% EXAMPLE
% [A,V0]=GGReadEdgeList('e01.txt','v01.txt',0);
```

C.2 Algorithms

```
% function VV= GCAFG(A,Scale)
% AFG community detection
%
% Front end for mscd_afg.m, which is E. le Martelot's implementation
% of the community detection method by Arenas, Fernandez, Gomez:
% A. Arenas, A. Fernandez, and S. Gomez. "Analysis of the
% structure of complex networks at different resolution levels."
% New Journal of Physics 10.5 (2008): 053039.
%
% INPUT
% A:      adjacency matrix of graph
% Scale:  a K-by-1 matrix of scale parameters (HIGH-TO-LOW values!!!)
%         every value yields a clustering
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Scale(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,13,3,0);
% VV=GCAFG(A,[3.0:-0.5:0.1]);
```

```
% function VV= GCDanon(A)
% Danon algorithm community detection
%
% Front end for danon.m, which is E. le Martelot's implementation
% of Danon's greedy community detection agglomerative method:
% L. Danon, A. Diaz-Guilera, A. Arenas. "The effect of size heterogeneity
% on community identification in complex networks." See Journal
% of Statistical Mechanics: Theory and Experiment 2006. 11 (2006): P11010.
%
% INPUT
% A:      adjacency matrix of graph
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Scale(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCDanon(A);
```

```
% function V=GCGloDens(A,Kmax,Iter)
% Community detection by global density maximization
%
% Graph clustering by stoch. optimization of global density.
% For details see the ComDet manual.
%
% INPUT
% A:      adjacency matrix of graph
% Kmax:   maximum number of clusters to consider
% Iter:   number of restarts of stochastic opt.
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         under the K-clusters clustering
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.85,0.2,0);
% VV=GCGloDens(A,5,5);
```

```
% function VV= GCHSLSW(A,Scale)
% HSLSW community detection
%
% This is a front end for mscd_hslsw.m, which is E. le Martelot's
% implementation of the method of Huang et al.:
% "Towards Online Multiresolution Community Detection in Large-Scale
% Networks",  PloS one 6.8 (2011): e23829.
%
% INPUT
% A:      adjacency matrix of graph
% Scale:  a K-by-1 matrix of scale parameters (HIGH-TO-LOW values!!!)
%         every value yields a clustering
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Scale(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCHSLSW(A,[3:-0.5:0.1]);
```

```
% function VV= GCLFK(A,Scale)
% LFK community detection
%
% This is a front end for mscd_lfk.m, which is E. le Martelot's
% implementation of the method of Lancichinetti, Fortunato, Kertesz:
% Lancichinetti, Andrea, Santo Fortunato, and János Kertész. "Detecting
% the overlapping and hierarchical community structure in complex
% networks." New Journal of Physics 11.3 (2009): 033015.
%
% INPUT
% A:      adjacency matrix of graph
% Scale:  a K-by-1 matrix of scale parameters (HIGH-TO-LOW values!!!)
%         every value yields a clustering
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Scale(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCLFK(A,[3:-0.5:0.1]);
```

```
% function V=GCLocDens(A,Kmax,Iter)
% Community detection by local density maximization
%
% Graph clustering by stoch. optimization of local  density.
% For details see the ComDet manual.
%
% INPUT
% A:      adjacency matrix of graph
% Kmax:   maximum number of clusters to consider
% Iter:   number of restarts of stochastic opt.
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         under the K-clusters clustering
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.85,0.2,0);
% VV=GCLocDens(A,5,5);
```

```
% function VV= GCModulMax1(A)
% Modularity Maximization community detection
%
% This is a front end for cluster_jl.m, A. Scherrer's implementation
% of the method of Blondel, Guillaume, Lambiotte and Lefebvre:
% "Fast unfolding of community hierarchies in large networks",
% http://arxiv.org/abs/0803.0476
%
% INPUT
% A:      Adjacency matrix of graph
%
% OUTPUT
% VV:     N-by-1 matrix, VV(n) is the cluster to which node n belongs
%
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,13,3,0);
% VV=GCModulMax1(A);
```

```
% function VV= GCModulMax2(A)
% Modularity Maximization community detection
% This is a front end for fast_mo.m, E. le Martelot's implementation
% of a fast greedy modularity maximization method
%
% INPUT
% A:      Adjacency matrix of graph
%
% OUTPUT
% VV:     N-ny-1 matrix, VV(n) is the cluster to which node n belongs
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,12,4,0);
% VV=GCModulMax2(A);
%
```

```
% function VV= GCModulMax3(A)
% Modularity Maximization community detection
% This is a front end for fast_newman.m, E. le Martelot's implementation
% of Newman's greedy agglomaerative modularity maximization method
% Newman, Mark EJ. "Fast algorithm for detecting community structure in
% networks." Physical review E 69.6 (2004): 066133.
%
% INPUT
% A:      Adjacency matrix of graph
%
% OUTPUT
% VV:     N-ny-1 matrix, VV(n) is teh cluster to which node n belongs
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCModulMax3(A);
%
```

```
% function V=GCNodeMemb(A,Kmax,Iter)
% Graph clustering by stoch. optimization of partition-based-based
% quality function. For details see the ComDet manual.
% This code is experimental and too slow for now (to be improved)
%
% INPUT
% A:      adjacency matrix of graph
% Kmax:   maximum number of clusters to consider
% Iter:   number of restarts of stochastic opt.
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Scale(k)
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.85,0.2,0);
% VV=GCNodeMemb(A,6,100)
```

```
% function VV= GCReichardt(A,Gamma)
% Reichardt community detection
%
% This is a front end for reichardt.m, which is E. le Martelot's
% implementation of gamma-modularity maximization. Gamma modularity
% as defined by Reichardt, Jorg, and Stefan Bornholdt. "Statistical
% mechanics of community detection." Physical Review E 74.1
% (2006): 016110.
%
% INPUT
% A:      adjacency matrix of graph
% Gamma:  a K-by-1 matrix of gamma values, each value yields a
%         a clustering
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n
%         belongs when algorithm uses Gamma(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCReichardt(A,[3:-0.5:0.1]);
```

```
% function VV= GCRonhovde(A,Gamma)
% Community detection using Ronhovde et al.'s method. This
% is a front end for ronhovde.m, which is E. le Martelot's
% implementation of gamma-modularity maximization, as defined by
% Ronhovde, Peter, and Zohar Nussinov. "Local resolution-limit-free
% Potts model for community detection." Physical Review E 81.4
% (2010): 046114.
%
% INPUT
% A:      adjacency matrix of graph
% Gamma:  a K-by-1 matrix of gamma values, each value yields a
%         a clustering
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Gamma(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCRonhovde(A,[0.5:-0.05:0.05]);
```

```
% function VV=GCSpectralClust1(A,Kmax)
%
% Community detection by spectral  clustering. See J. Hespanha's
% implementation of spectral clustering. For details see
% Joao Hespanha. "An efficient MATLAB Algorithm for Graph Partitioning".
% Technical Report, University of California, Oct. 2004.
% http://www.ece.ucsb.edu/~hespanha/techrep.html.
%
% INPUT
% A:      adjacency matrix of graph
% Kmax:   max number of clusters  to consider. A clustering of K clusters
%         will be produced for every K in [1:Kmax]
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses a partition of k clusters
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,13,3,0);
% VV=GCSpectralClust1(A,6)
```

```
% function VV=GCSpectralClust2(A,Kmax,T)
% Community detection by spectral  clustering
%
% Community detection by spectral  clustering. See for example
% "J. Shi and J. Malik. Normalized cuts and image segmentation.
% IEEE Transactions on Pattern Analysis and Machine Intelligence
% 22(8):888–905, 2000."
%
% INPUT
% A:      adjacency matrix of graph
% Kmax:   maximum number of clusters to consider
% T:      Number of times to repeat the k-means clustering, each
%         with a new set of initial cluster centroid positions
%         (option for MATLAB Statistics toolbox kmeans function)
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n
%         belongswhen algorithm uses Gamma(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,13,3,0);
% VV=GCSpectralClust2(A,6,10)
```

```
% function VV=GCStabilityOpt(A,Scale)
% Graph clustering by stability optimization.
%
% This is a front end for mscd_so.m, which is E. le Martelot's
% implementation of Le Martelot and  Hankin's stability optimization
% method. See "Multi-scale Community Detection using Stability as
% Optimisation Criterion in a Greedy Algorithm, Proceedings of
% KDIR 2011".
%
% INPUT
% A:      adjacency matrix of graph
% Scale:  a K-by-1 matrix of scale parameters (LOW-TO-HIGH VALUES!!!);
%         every value yields a clustering
%
% OUTPUT
% VV:     N-ny-K matrix, VV(n,k) is the cluster to which node n belongs
%         when algorithm uses Scale(k)
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,13,3,0);
% VV=GCStabilityOpt(A,[0.1:0.4:2.5])
```

C.3 Cluster Number Selection

```
% function Kbst=CNDistBased(VV,A)
% Partition Distance based cluster number selection
%
% Cluster number selection performed by finding the VV column which
% achieves  highest value of the clustering  quality function QFDB
% (see documentation in Evaluation/QFDB.m)
%
% INPUT
% VV:     N-by-K matrix of partitions, k-th column describes a partition
%         of k clusters
% A:      adjacency matrix of graph
%
% OUTPUT
% Kbst:   the number of best VV column and so best number of clusters
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNDistBased(VV,A);
```

```
% function Kbst = CNFixed(VV,K)
% Cluster Number Selection by fixed number K
%
% Cluster Number Selection: let selected Kbst equal given K
% This function simply returns the cluster number which is
% input by the user
%
% INPUT
% VV:     N-by-K matrix of partitions, k-th column describes a partition
%         of k clusters
% K:      desired number of clusters
%
% OUTPUT
% Kbst:   K, the input number
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCAFG(A,[0.2:0.5:2.5]);
% Kbst=CNFixed(4);
% V=VV(:,Kbst);
% Q=PSRelCluNumError(V,V0)
% Q=PSNMI(V,V0)
```

```
% function Kbst=CNGloDens(VV,A)
% Global-density-based cluster number selection
%
% Cluster number selection performed by finding the VV column which
% achieves  highest value of the global density clustering  quality
% function QFGloDens (see documentation in Evaluation/QFGD.m)
%
% INPUT
% VV:     N-by-K matrix of partitions, k-th column describes a partition
%         of k clusters
% A:      adjacency matrix of graph
%
% OUTPUT
% Kbst:   the number of best VV column and so best number of clusters
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNGloDens(VV,A);
```

```
% function Kbst=CNLocDens(VV,A)
% Local-density-based cluster number selection
%
% Cluster number selection performed by finding the VV column which
% achieves  highest value of the local density clustering  quality
% function QFLD (see documentation in Evaluation/QFLD.m)
%
% INPUT
% VV:     N-by-K matrix of partitions, k-th column describes a partition
%         of k clusters
% A:      adjacency matrix of graph
%
% OUTPUT
% Kbst:   the number of best VV column and so best number of clusters
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNLocDens(VV,A);
```

```
% function Kbst=CNModul(VV,A)
% Modularity based cluster number selection
%
% Cluster number selection performed by finding the VV column which
% achieves  highest Newman-Girvan modularity (see Evaluation/QFModul.m)
%
% INPUT
% VV:     N-by-K matrix of partitions, k-th column describes a partition
%         of k clusters
% A:      adjacency matrix of graph
%
% OUTPUT
% Kbst:   the number of best VV column and so best number of clusters
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,16,0,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
```

```
% function Kbst=CNNodMemb(VV,A,m)
% Cluster Number Selection by node membership
%
% Cluster number selection performed by finding the VV column which
% achieves  highest value of the node membership clustering  quality
% function QFNM (see documentation in Evaluation/QFNM.m)
%
% INPUT
% VV:     N-by-K matrix of partitions, k-th column describes a partition
%         of k clusters
% A:      adjacency matrix of graph
% m:      internal variable, always set to 1
%
% OUTPUT
% Kbst:   the number of best VV column and so best number of clusters
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNNodMemb(VV,A,1);
```

C.4 Evaluation

```
% function Q=PSPSRelCluNumError(V,V0)
% Relative cluster number error
%
% Relative cluster number error: abs(K-K0)/K0
% Becomes zero when V has the same number of clusters as V0
%
% INPUT
% V:        N-by-1 matrix describes test partition
% V0:        N-by-1 matrix describes reference partition
%
% OUTPUT
% Q:         The relative Cluster Number error of  V wrt V0
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,12,4,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
% V=VV(:,Kbst);
```

```
% function Q=PSJaccard(V,V0)
% Jaccard index
%
% Computes the Jaccard index, shows similarity between partitions
% V and V0. Max similarity is 1 and min similarity is 0. See
% http://en.wikipedia.org/wiki/Jaccard_index
%
% INPUT
% V:        N-by-1 matrix describes 1st partition
% V0:        N-by-1 matrix describes 2nd partition
%
% OUTPUT
% Q:         The Jaccard similarity between V and V0
%
% EXAMPLE
% [A,V0]=GGGN(32,4,16,0,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
% V=VV(:,Kbst);
% Q=PSJaccard(V,V0);
%
```

```
% function Q=PSNMI(V,V0)
% Normalized Mutual Information (NMI)
% An implementation of Normalized Mutual Information (NMI)
% by Erwan Le Martelot. The NMI measure shows
% the similarity between two partitions. Max similarity is 1
% and min similarity is 0. For details see Danon, Leon, et al.
% "Comparing community structure identification." Journal of
% Statistical Mechanics: Theory and Experiment 2005.09 (2005): P09008.
%
% INPUT
% V:        N-by-1 matrix describes 1st partition
% V0:        N-by-1 matrix describes 2nd partition
%
% OUTPUT
% Q:         The Normalized Mututal Information between V and V0
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,12,4,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
% V=VV(:,Kbst);
% Q=PSNMI(V,V0);
%
```

```
% function Q=PSRand(V,V0)
% Rand index
%
% Computes the Rand index, which shows similarity between two partitions
% V and V0. Max similarity is 1 and min similarity is 0. See
% http://en.wikipedia.org/wiki/Rand_index
%
% INPUT
% V:        N-by-1 matrix describes 1st partition
% V0:        N-by-1 matrix describes 2nd partition
%
% OUTPUT
% Q:         The Rand similarity between V and V0
%
% EXAMPLE
% [A,V0]=GGGN(32,4,16,0,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
% V=VV(:,Kbst);
% Q=PSRand(V,V0)
%
```

```
% function Q=PSPSRelCluNumError(V,V0)
% Relative cluster number error: abs(K-K0)/K0
% Becomes zero when V has the same number of clusters as V0
%
% INPUT
% V:        N-by-1 matrix describes test partition
% V0:        N-by-1 matrix describes reference partition
%
% OUTPUT
% Q:         The relative Cluster Number error of  V wrt V0
%
% EXAMPLE
% [A,V0]=GGGirvanNewman(32,4,12,4,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
% V=VV(:,Kbst);
% Q=PSRelCluNumError(V,V0);
%
```

```
%function Q=QFDistBased(V,A)
% A partition-distance-based quality function
%
% A partition-distance-based quality function
% For more details see the ComDet Toolbox manual
%
% INPUT
% V:      N-by-1 matrix describes a partition
% A:      adjacency matrix of graph
%
% OUTPUT
% Q:      node-membership-based quality function of V given graph
%         with adj. matrix A
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCDanon(A);
% Kbst=CNNM(VV,A);
% V=VV(:,Kbst);
% Q=QFDistBased(V,A)
```

```
% function Q=QFGloDens(V,A)
% A global-density-based quality function
%
% A global-density-based quality function
% For more details see the ComDet Toolbox manual
%
% INPUT
% V:      N-by-1 matrix describes a partition
% A:      adjacency matrix of graph
%
% OUTPUT
% Q:      local-density-based quality function of V given graph
%         with adj. matrix A
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCDanon(A);
% Kbst=CNGloDens(VV,A);
% V=VV(:,Kbst);
% Q=QFGloDens(V,A)
```

```
% function Q=QFLocDens(V,A)
% A local-density-based quality function
%
% A local-density-based quality function
% For more details see the ComDet Toolbox manual
%
% INPUT
% V:      N-by-1 matrix describes a partition
% A:      adjacency matrix of graph
%
% OUTPUT
% Q:      node-membership-based quality function of V given graph
%         with adj. matrix A
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCDanon(A);
% Kbst=CNNM(VV,A);
% V=VV(:,Kbst);
% Q=QFLocDens(V,A)
```

```
% function Q = QFModul(V,A)
% Modularity  quality function
%
% Computes the classical Newman-Girvan modularity. The code for
% its evaluation, listed below, was written by E. le Martelot.
% See http://en.wikipedia.org/wiki/Modularity_%28networks%29
%
% INPUT
% V:      N-by-1 matrix describes a partition
% A:      adjacency matrix of graph
%
% OUTPUT
% Q:      the modularity of V given graph (with adj. matrix) A
%
% EXAMPLE
% [A,V0]=GGGN(32,4,16,0,0);
% VV=GCAFG(A,[0.2:0.5:1.5]);
% Kbst=CNModul(VV,A);
% V=VV(:,Kbst);
% Q = QFModul(V,A)
%
```

```
% function Q=QFNodMemb(V,A)
% A node-membership-based quality function
%
% A node-membership-based quality function
% For more details see the ComDet Toolbox manual
%
% INPUT
% V:      N-by-1 matrix describes a partition
% A:      adjacency matrix of graph
%
% OUTPUT
% Q:      node-membership-based quality function of V given graph
%         with adj. matrix A
%
% EXAMPLE
% [A,V0]=GGPlantedPartition([0 10 20 30 40],0.9,0.1,0);
% VV=GCDanon(A);
% Kbst=CNNodMemb(VV,A);
% V=VV(:,Kbst);
% Q=QFNodMemb(V,A)
```