Chinese Whispers User Manual

Chris Biemann
Version 1.0, May 2006

Table of Contents

Introduction

Chinese Whispers, henceforth abbreviated CW, is an efficient graph clustering algorithm. It has been described first in [Biemann and Teresniak 2005] and more elaborately in [Biemann 2006].
CW is a clustering algorithm that clusters undirected, weighted graphs. Its run-time complexity is linear in the number of edges, which makes CW in comparison a very efficient algorithm. The output is a fuzzy partitioning (non-hierarchical) of the graph.

The reader is referred to the papers mentioned above for motivation, detailed algorithm description and possible application areas of CW. This manual merely contains instructions how to use the Graphical User Interface (GUI) and the command line version of CW and provides only short, informal descriptions of the algorithm.

The CW program is licensed under the Gnu Public License, see http://www.gnu.org/licenses/gpl.html.

Installation

The implementation is written in JAVA and platform independent. It was tested on MS Windows, Linux and MacOS X.
Download the archive http://wortschatz.uni-leipzig.de/~cbiemann/software/CW.zip into a folder of your choice.

System requirements

To run CW, you need a Java Runtime Environment (JRE) of version 1.5 or later. You can obtain it at http://java.sun.com/j2se/1.5.0/download.jsp. Please ensure that *java* is in the path – to check this, type "*java -version*" in your shell – it should respond with a version number of 1.5.0 or higher.
If you want to use a database to store your graphs and the clustering results, it is recommended that you download and install MySQL from http://dev.mysql.com/downloads/ . However, if you do not use a database anyway, this might be too much trouble, as CW does not depend on a database.

Running the program

Change to the directory you unpacked the archive to and type

java –jar CW.jar

in your command line window. The GUI will start up, provided that java is in your path.
If you process large graphs, however, it is recommended to allow CW to use more memory. You can specify the maximum amount of memory (here 256 Megabytes) like

java –Xmx256M –jar CW.jar

Windows users might use the script *runCW.bat*, for UNIX-based operating systems, the script *runCW.sh* will call CW.

Experimenting with CW

This section deals with how to operate CW. In section 3.1, a short introduction on how CW works is given. It is recommended to first get acquainted with the GUI (described in section 3.2) to get a feeling for what CW does and what the effects of the different options are. If you want to use CW in batch mode, section 3.3 tells you how to specify the command line options.

How CW works

In this section the CW algorithm is outlined roughly. Please consider [Biemann 2006] for details. CW is a very basic – yet effective – algorithm to partition the nodes of weighted, undirected graphs. It is motivated by the eponymous children's game, where children whisper words to each other. While the game's goal is to arrive at some funny derivative of the original message by passing it through several noisy channels, the CW algorithm aims at finding groups of nodes that broadcast the same message to their neighbours.

The algorithm is outlined like this:

```
initialize:
 forall v_i in V: class(v_i)=i;
while changes:
 forall v in V, randomized order:
   class(v)=highest ranked class
            in neighbourhood of v;
```

Intuitively, the algorithm works as follows in a bottom-up fashion: First, all nodes get different classes. Then the nodes are processed for a small number of iterations and inherit the strongest class in the local neighbourhood in an update step. The strongest class is the class whose sum of edge weights to the current node is maximal. In case of multiple strongest classes, one is chosen randomly. Regions of the same class stabilize during the iteration and grow until they reach the border of a stable region of another class. Note that classes are updated immediately: a node can obtain classes from the neighborhood that were introduced there in the same iteration.
The CW algorithm cannot cross component boundaries, because there are no edges between nodes belonging to different components. Further, nodes that are not connected by any edge are discarded from the clustering process, which possibly leaves a portion of nodes unclustered.
Apart from ties, the classes usually do not change any more after a handful of iterations. The number of iterations depends on the diameter of the graph.
The result of CW is a hard partitioning of the given graph into a number of partitions that emerges in the process – CW is parameter-free (yet a lot of options can be chosen in this implementation). It is possible to obtain a soft partitioning by assigning a class distribution to each node, based on the weighted distribution of (hard) classes in its neighbourhood in a final step. The result of CW is non-deterministic, i.e. the clustering the same graph several times can result in different outcomes.

A graph is specified by a list of nodes and a list of edges. Nodes have IDs and labels, edges consist of two node ids and an edge weight. For example, a triangle graph with nodes A B and C and edge weights A-B =10, A-C =20, B-C = 30 is encoded like this:

Node List:

1	A
2	B
3	C

Edge list:

1	2	10
2	1	10
1	3	20
3	1	20
2	3	30
3	2	30

Please note that the edge list is symmetric. CW will run for non-symmetrical edge lists of edge lists that specify different weights for different directions of the same edge, but this is unexplored and not recommended.
When reading the graph from files, the columns (as in the triangle example) are tab-separated. See the folder “examples” for sample graphs. When reading the graph from a database, the

columns are to be specified in the database panel. Note that the edge list must have an additional primary key (see database scheme).

For the moment, edge weights are positive integer values. A version that accepts floating point numbers as weights is planned in the near future.

Options

Different options can be set to change the outcome of CW. The algorithm option can be set to four values: *top*, *dist log*, *dist nolog* and *vote* (with a threshold). The difference is illustrated in figure 1: Here, the centre node A changes its label in an update step differently, according to the option chosen, which influences the ranking of the classes in the neighbourhood of nodes: *top* sums over the neighbourhood's classes, *dist* downgrades the influence of a neighbouring node by its degree (number of edges of node) and *vote* is the same as *top* but needs a minimum fraction (this is the numerical parameter) for a class change to take place.

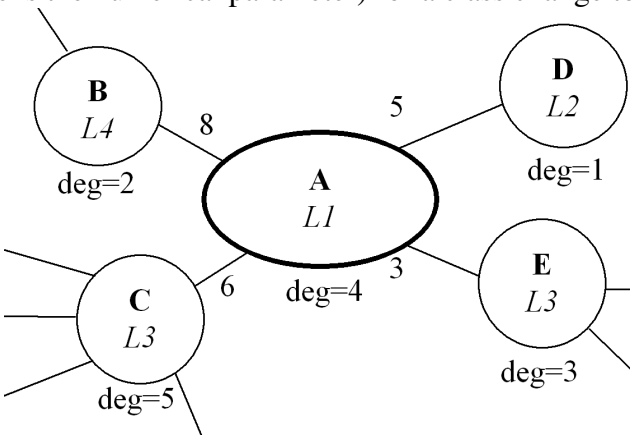

Figure 1: An update situation for node A

The strengths of classes for the situation in figure 1 are, dependent on the algorithm option:

top:	strength(**L3**)=9; strength (L4)=8; strength (L2)=5
dist nolog:	strength (**L2**)=5; strength (L4)=4; strength (L3)=2.2
dist log:	strength (**L4**)=7.28; strength (L3)=5.51; strength (L2)=3.46
vote:	strength (**L3**)=0.409; strength (L4)=0.363; strength (L2)=0.227

If the vote threshold is set to a value above 0.409, then A keeps its label L1.

Other options are:

- mutation option and numerical value: With some probability, a node changes to a yet unused class. This mutation rate can be exponentially decreasing in the iteration or constant. The numerical parameter is the speed of decrease and the mutation rate in the constant case, respectively.
- keep class rate: With this probability, a node is not updated but keeps its class until the next iteration. If e.g. this parameter is set to 0.1, then in average in a 20 nodes graph, 2 nodes do not perform any update step
- update strategy: stepwise vs. constant. In stepwise mode, the updates are not performed immediately, but take effect in the next iteration. In continuous update mode, a node can spread its class in the same iteration as it received it. The continuous option converges faster.

The default setting for CW is option *top*, keep class rate 0, mutation constant with rate 0 (no mutation) and continuous update.

If stepwise update is chosen and neither mutation nor keep class rate is switched on, the algorithm might converge to a set of oscillating states, see [Biemann 2006].

Operating the GUI

After starting CW, you will see a welcome panel as shown in figure 2:

Figure 2: Welcome Panel

In the upper part you can chose between this and three other panels. The second panel is the main panel as shown in figure 3:

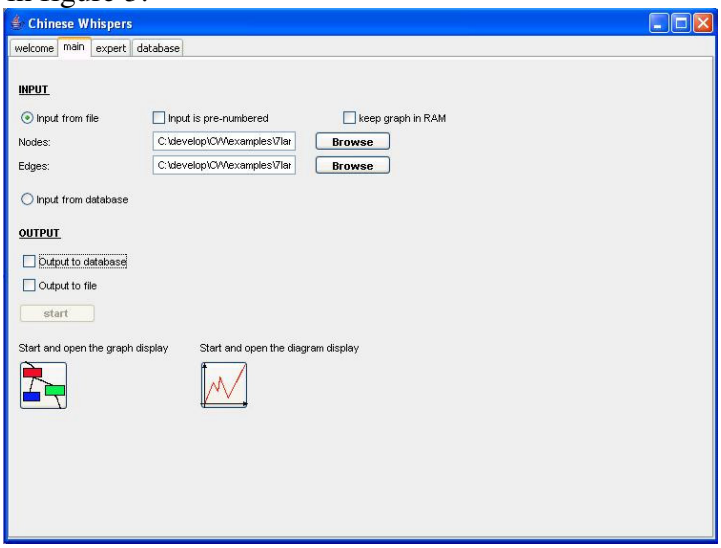

Figure 3: Main Panel

Input

In the main panel, the source of the graph to be clustered and the destination of the output can be specified. For input, you can choose between files and database.

The options “keep graph in RAM” should be unchecked only if you use graphs that do not fit into your machine’s main memory. Selecting the option “Input is pre-numbered” does not compress the temporary binary files your graph will be encoded to, but saves time and RAM for large graphs.

For output, you can decide between either writing the output to files or to a database, or displaying the outcome of the algorithm in an interactive graph window or in a diagram that shows quantitative characteristics of the result.

For the moment, chose “`20nodes.txt`” in the folder “`examples`” as node list and “`20edges.txt`” in the same folder as edge list.

Output as graph

Press the button for the graph display. A new windows should open up, looking like figure 4:

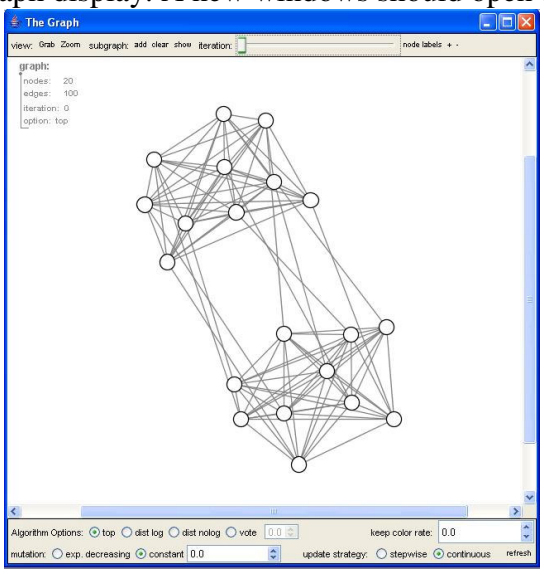

Figure 4: A sample graph with 20 nodes

Now pull the slider next to “Iteration” to the right. You will follow the different clusterings in increasing iterations of CW. This could look like in figure 5:

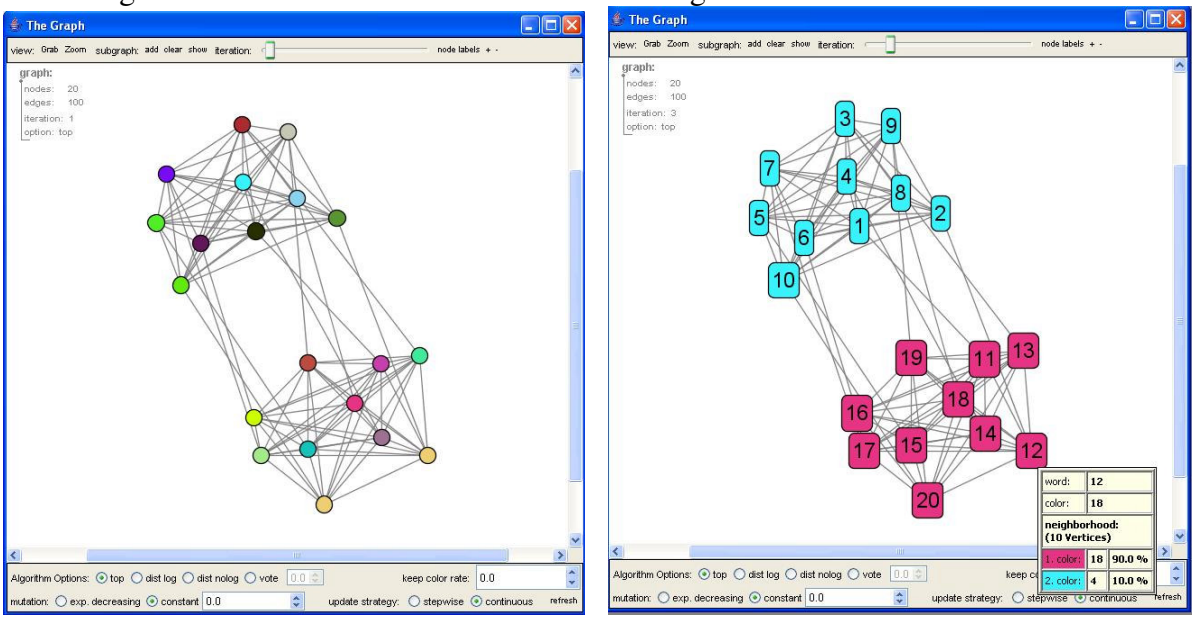

Figure 5: different iterations for the 20 nodes sample graph

Different colours represent different classes. If you move the mouse over a node (cf. fig. 4, lower right corner), you get information on to what extent the node belongs to different classes. For switching between labelled and unlabeled nodes, use the button “node labels” in the upper controls. You can also change the size of the nodes using “+” and “-“.

If you use a wheel mouse, you can zoom in and out using the wheel. For all other users, a zoom window is provided (click on “zoom”) that allows you to broaden or to narrow your view on the graph. Figure 6 illustrates the zoom window.

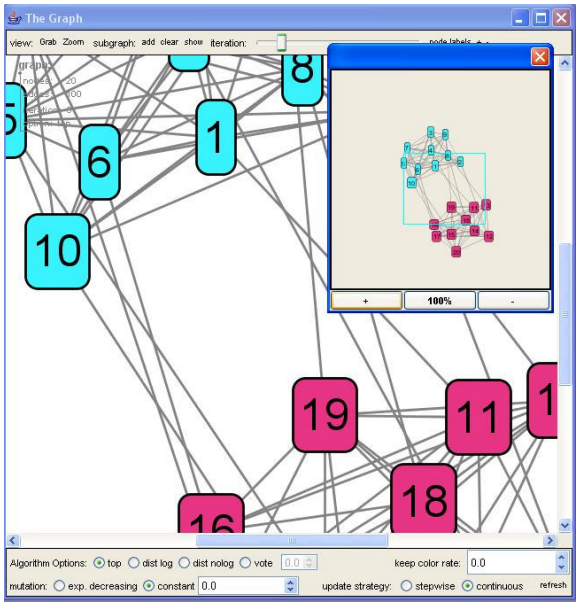

Figure 6: Zoom window

For graphs with more than a couple of hundred nodes, the graph display becomes slow and not laid out good enough to reveal any structure. To be able to still view the graph, a sub graph view was implemented that only displays the neighbourhood of some specified nodes of particular interest.

For the following couple of figures, the 7lang-graph as available in the examples was used. (`7lang_nodes.txt` and `7lang_edges.txt`). This graph comprises the significant co-occurrences of a corpus compiled from seven different languages. CW clusters the words according to the language, see [Biemann and Teresniak 2005].

Figure 7 shows how to specify a node for the sub graph by clicking on “add” in the graph window. Here, we entered “de”, the label of a node. It is possible to add several nodes. As soon as “show” is clicked, only the sub graph as induced by the neighbourhoods of the specified nodes (including these nodes) is drawn. Alternatively, you can add nodes to the sub graph by double-clicking on them.

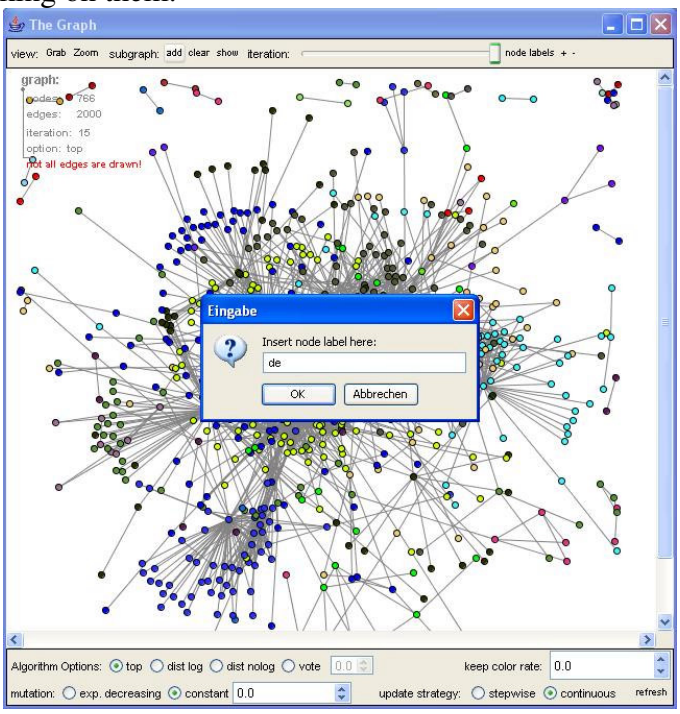

Figure 7: Adding a node to the sub graph

Figure 8 shows the sub graph for “de”. In the sub graph, you can add further nodes or return to the overall graph by using the “close”-button.

The lower part of the graph window allows to change algorithm parameters. The “refresh”-button results in re-calculation of the algorithm without changing the graph layout.

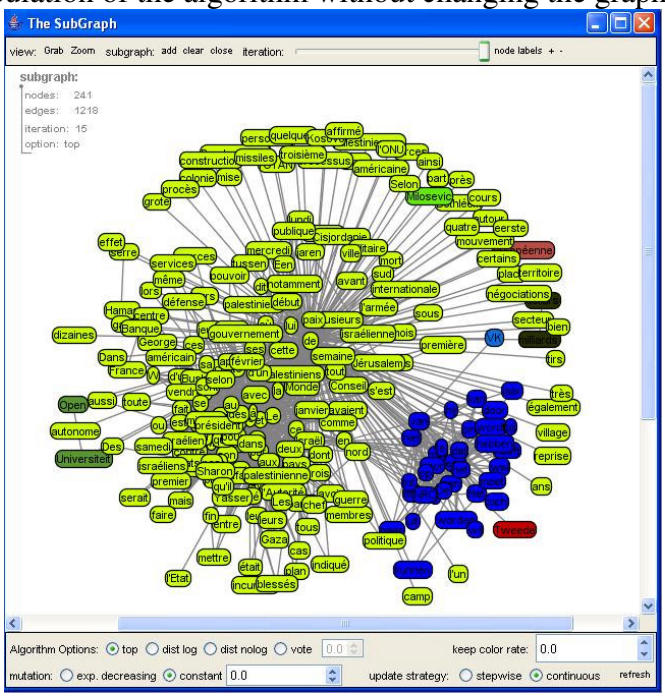

Figure 8: The sub graph for “de”.

Output as Diagram

Another possibility of displaying the result of CW is to draw the distribution of cluster sizes in a graph. This happens when using the graph button in the main panel (figure 3). A diagram window opens, figure 9 shows the diagram for the 7-lang graph.

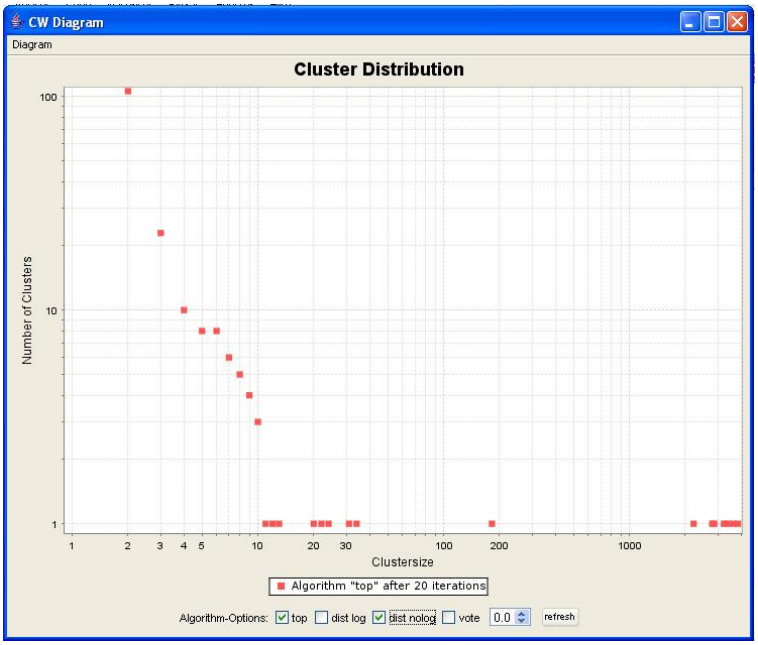

Figure 9: Diagram for the 7-lang graph

From figure 9 it is visible that there are a couple of large clusters (lower right corner), some medium-sized clusters and a large number of small clusters. This cluster distribution, especially the linear progression in this log-log-scale plot, is typical for applying CW on so-called small world graphs.
It is possible to enable other algorithm options in order to compare them and to save the diagram in picture format. Figure 10 displays what happens after enabling all options and pressing the "refresh"-button. Note that all parameters not adjustable in the diagram window (such as mutation, update strategy, keep class rate, minimum edge weight and number if iterations) stay the same.

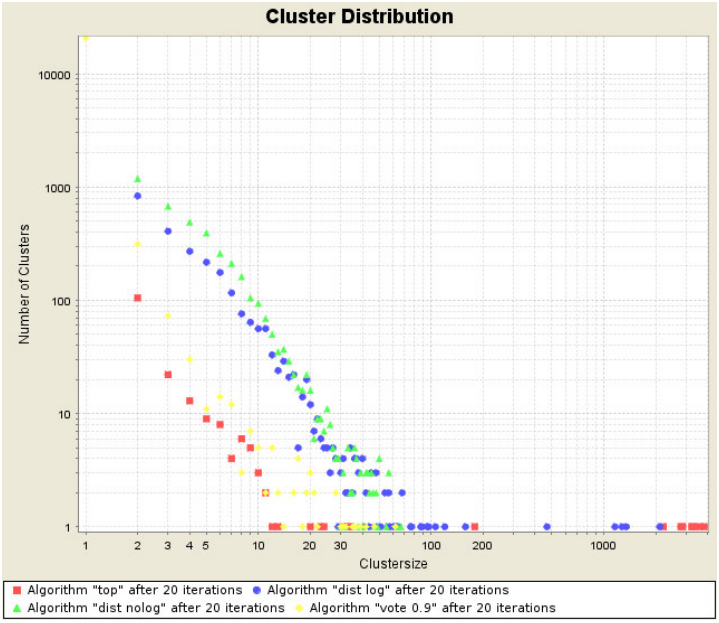

Figure 10: many runs in one diagram

Note that in comparison to the *top* option, the *dist log* option produces more and smaller clusters. The *dist nolog* option results in even more and even smaller clusters breaking the larg clusters apart. Using *vote 0.9* here leads to many classes with only one member (upper left corner of diagram).

Output in File
The output of CW can not only be displayed as a graph or as a diagram, it can also be written to a file or to a database. When checking "write to file" in the main panel and then pressing "start", a file dialogue will open after the clustering has been finished and two files will be written: A file including soft clustering information and a file with the extension .read that contains the classes in a readable format, but without soft clustering information.

For the 20-nodes sample graph, the result file looks like this:

node ID	node label	class ID	class 1 ID	class 1 %	class2 ID	class 2 %
1	1	1	1	90.0	2	10.0
2	2	1	1	90.0	2	10.0
3	3	1	1	90.0	2	10.0
4	4	1	1	90.0	2	10.0
5	5	1	1	90.0	2	10.0
6	6	1	1	90.0	2	10.0
7	7	1	1	90.0	2	10.0
8	8	1	1	90.0	2	10.0
9	9	1	1	90.0	2	10.0
10	10	1	1	90.0	2	10.0

11	11	2	2	90.0	1	10.0
12	12	2	2	90.0	1	10.0
13	13	2	2	90.0	1	10.0
14	14	2	2	90.0	1	10.0
15	15	2	2	90.0	1	10.0
16	16	2	2	90.0	1	10.0
17	17	2	2	90.0	1	10.0
18	18	2	2	90.0	1	10.0
19	19	2	2	90.0	1	10.0
20	20	2	2	90.0	1	10.0

The columns contain the following information:

- node ID: the ID number of the node
- node label: the corresponding label. In this example, ID and label are equal
- class ID: the class from clustering. Here, the graph is clustered into two parts
- class 1 ID: the prominent class in the neighbourhood
- class 1 %: The strength of class 1 in %
- class 2 ID: the second prominent class in the neighbourhood
- class 2 %: The strength of class 2 in %

The read file could have this content:

```
classID     # members     labels
1           10            1, 2, 3, 4, 5, 6, 7, 8, 9, 10
2           10            11, 12, 13, 14, 15, 16, 17, 18, 19, 20
```

This means that class 1 (col. 1) has 10 members (col. 2) and their labels are 1, 2, 3, 4, 5, 6, 7, 8, 9 and 10. Class 2 has also 10 members with labels 11, 12, 13, 14, 15, 16, 17, 18, 19, 20.

In the expert panel (figure 11), all the parameters can be specified in the “ALGORITHM” section. Additionally, the minimum edge weight threshold can be set that leads to ignoring edges that fall short this threshold. This is effectively the main parameter, for CW. Further, the number if iterations can be set. Changes do only take effect if the “apply”-Button below is pressed.

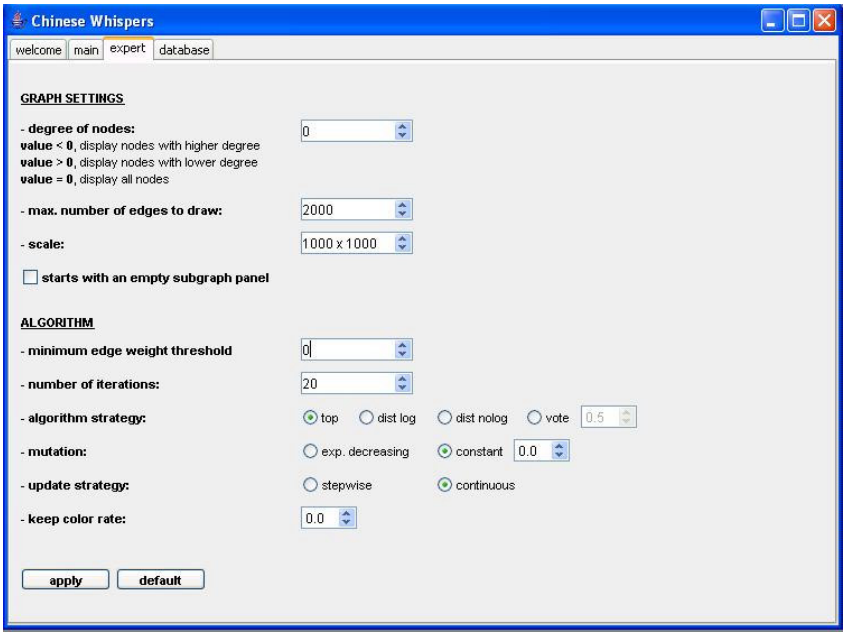

Figure 11: The expert panel

Changes in the "GRAPH SETTINGS" section only affect the display of the graph in the graph window, not the algorithm itself. Setting the degree of nodes to a positive value, say 5, will only display nodes that have a lower degree than 5, therefore omitting so-called "hubs". This often leads to a nicer layout of the graph. To see only the highly connected nodes of a degree more than say, 20, set the degree of nodes to the negative value of –20.
The maximum number of edges to draw is a means not to slow down the process of graph drawing too much by painting only the edges with the highest weight. The scale changes the granularity of the display.

Using a Database
If you want to use a database for your graphs and/or for your results, the database panel as depicted in figure 12 provides the possibility to provide the necessary information. Note that the fields can only be edited if the corresponding source/destination has been chosen in the main panel.
In the database settings, the database is specified. The program was only tested with the MySQL 5.0 database server so far. See http://dev.mysql.com/doc/mysql/en/ for details and http://dev.mysql.com/downloads/ for downloads.
This manual is not meant to introduce you to databases, so if you do not use databases anyway, you can safely skip the remainder of the section.
The graph should be provided in two tables, similar to the file input format. Tables have names and columns, which can be set in the respective input fields. Please note that for the reason of being able to handle large graphs, edge tables must have an additional numerical primary key column (which is found automatically). For illustration, a sample database with **some graphs** is included in the distribution in the folder `db_cw`. To make this database available to your MySQL system, simply copy the folder and all its contents into the data directory (e.g. C:\Programs\MySQL\MySQL Server 5.0\data).
The output table will be created or overwritten and contains the same information as the result file.

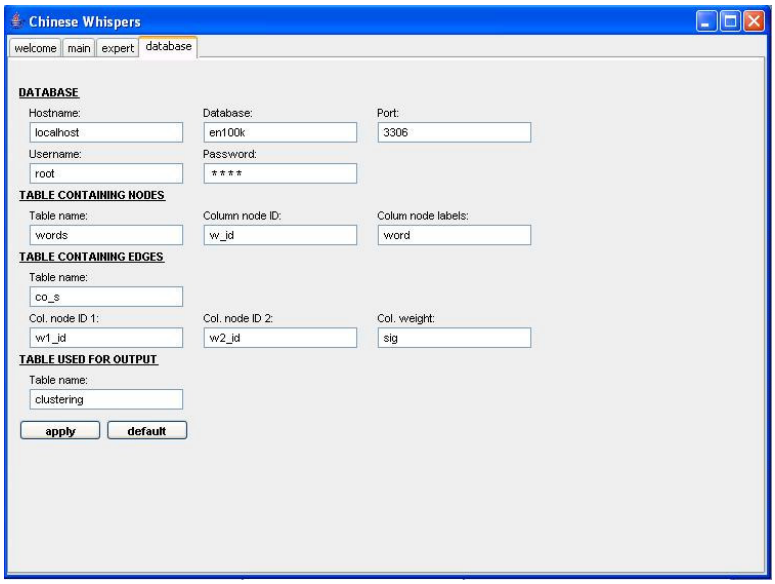

Figure 12: database panel

Operating the command line version

All settings that can be undertaken in the GUI can also be applied in a CW command line run. This is meant for people that are already familiar with CW and want to use it as part of a system in batch mode or do not have a graphical display.

To see the command line options, start CW with a command line parameter, e.g.

java –jar CW.jar –h.

You will see the following text in your terminal:

```
-H | -h | --help   Writes out this Help.
-D Use database specified in CW_DBproperties.ini as input.
-F Use files specified by -i as input.
-i Use files as input.
 filename1  The node list 2-col.
 filename2  The edge list 3-col.
-a Sets the algorithm options
 "top"
 "dist_nolog"
 "dist_log"
 "vote x" with x in [0.0,1.0]
-t Weight threshold (default 0)
-k Keep class rate (default 0.0)
-m Mutation mode [dec|constant] value(pos.real)
-d Number of iterations x>0 (default x=20).
-o Writes clustering to filename.
 filename    The filename for output.
-O Writes clustering into database specified in CW_DBproperties.ini.
-S Do not renumber input.
-R keep graph on disk (large graphs).
```

This should enable you to specify the options as described in the GUI section.
For example, to cluster the sample 20-nodes graph with algorithm option “dist log”, keep class rate 0.44, constant mutation rate of 0.21, using 17 iterations and writing the output to a result table as specified in `CW_DBproperties.ini`, type the following:

java –jar CW.jar -F -i examples/20nodes.txt examples/20edges.txt -a dist_log -k 0.44 -m constant 0.21 -d 17 -O

For clustering the 7lang-graph with default CW options into a result file named 7lang_result.txt, use

java –jar CW.jar -F -i examples\7lang_nodes.txt examples\7lang_nodes.txt -o 7lang_result.txt

Acknowledgements

CW was developed by Chris Biemann at the University of Leipzig. The GUI was implemented by Sebastian Gottwald, Rocco Gwizdziel and Chris Biemann. Thanks goes to all the testers from the NLP Department, University of Leipzig. Further, thanks goes to Vincenzo Moscati for the name of the algorithm. Special thanks goes to Uwe Quasthoff for many fruitful discussions.

References

[Biemann 2006] Biemann, C. (2006): Chinese Whispers - an Efficient Graph Clustering Algorithm and its Application to Natural Language Processing Problems. Proceedings of the HLT-NAACL-06 Workshop on Textgraphs-06, New York, USA. http://wortschatz.uni-leipzig.de/~cbiemann/pub/2006/BiemannTextGraph06.pdf

[Biemann and Teresniak 2005] Biemann, C., Teresniak, S. (2005): Disentangling from Babylonian Confusion - Unsupervised Language Identification, Proceedings of CICLing-2005, Computational Linguistics and Intelligent Text Processing, Mexico City, Mexico and Springer LNCS 3406
http://wortschatz.uni-leipzig.de/~cbiemann/pub/2005/cicling05.pdf