MotifAnalysis: Guide

Stefanie Roos
roos@cs.tu-darmstadt.de

September 29, 2011

1 Introduction

MotifAnalysis is a Java-program designed for analyzing motifs, i.e., induced subgraphs, in networks. Functions for generating diagrams of various results are integrated. However, they require that the plotting software 'Gnuplot' is installed.

Currently only the analysis of undirected graphs and motifs up to 4 nodes is implemented. For each motif m the following single values can be computed:

- the number of m-instances in the graph
- the percentage of m-instances with respect to all considered motifs
- in presence of edge weights, respectively node weights:
  - total weight of all m-instances
  - average weight per instance
  - average weight per edge respectively node
  - percentage of weight of m-instances with respect to all considered motifs

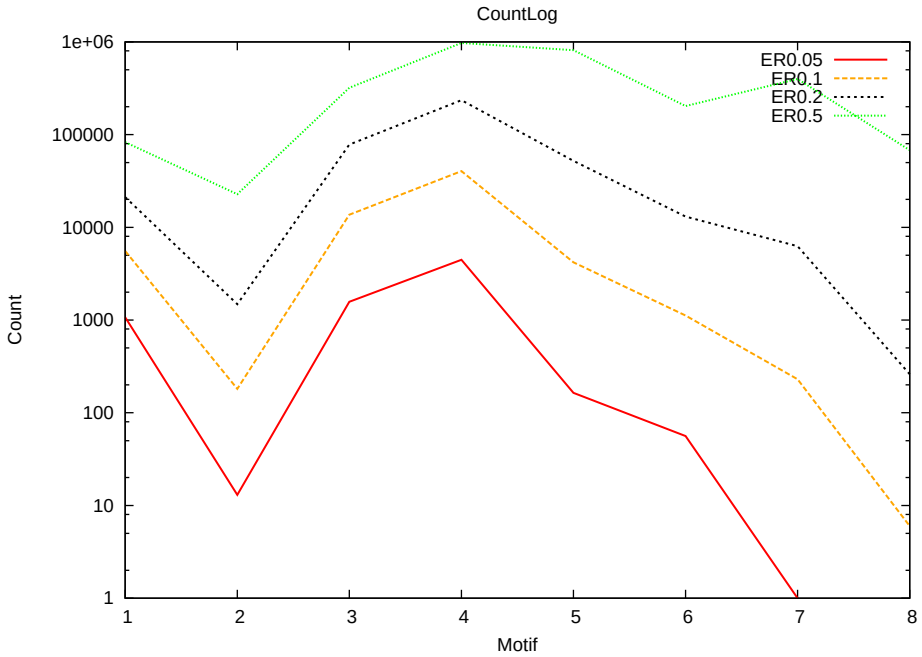

Figure 1: Motif Counts of Erdös-Renyi graphs

The above analysis can also be executed only for a certain node n, i.e., all instances n is part of are analyzed. Additionally the following distributions can be computed. In case of weight distributions the presence of integer weights is necessary.

- node id vs. number/edge weight/node weight of m-instances node is part of
- node id vs. fraction of (weight of) m-instances of all motifs n is part of
- edge weight/node weight vs. number of instances with this weight (in numbers, fraction, cumulative distribution function)

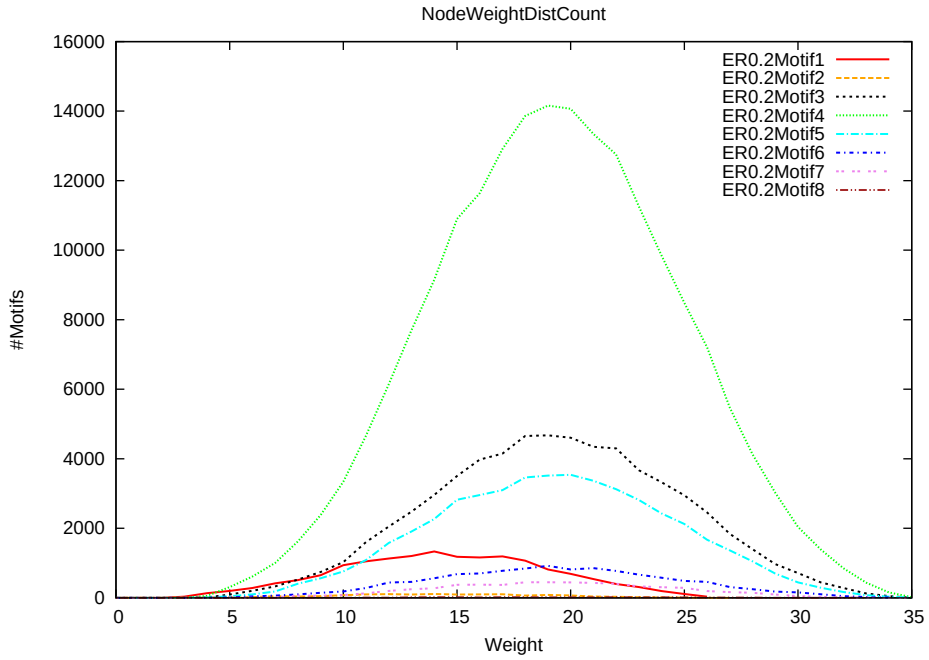

Figure 2: Node Weight Distribution of an Erdös-Renyi graph

- motif count/edge weight/node weight vs. number of nodes with this count/weight (in numbers, fraction, cumulative distribution function)

It is also possibly to write all instances, respectively a predefined number in case only a sample is of interest, of m to file.

For visualization the software offers an 2D-plotting component for the above singles and distributions. While singles are in general plotted in 'Motif Nr vs. Value' comparing various graphs, distributions are plotted as 'Value X vs Value Y' for each motif, offering the possibility to either compare one motif for various graphs or various motifs for one graph (or any combination of both if this should be of interest). Plots can be done with a logarithmic scale on the y-axis. Figures 1-3 show examples of the plotting capabilities. 4 Erdös-Renyi graphs were generated with 100 nodes each and parameter p, i.e. the probability that two randomly selected nodes are linked, chosen as 0.05,0.1,0.2,0.5. Edge and node weights were chosen uniformly at random between 1 and 10.

Figure 1 shows the number of each of the 8 undirected 3/4-node motifs, using a logarithmic y axis. Figure 2 shows the distribution of a node weights over all instances considering the 8 motifs for the ER graph with $p = 0.2$. Figure 3 is a scatter plot. The x axis displays the index of the node while the y axis shows the number of motif 2, i.e. clique of size 3, for each of the 4 graphs.

The remaining documentation is divided into two parts. Section 2 gives a brief introduction on how to use the software, without going into details on the actual implementation. For those interested in understanding and extending the software, Section 3 offers a brief introduction.

2 Basic Usage

This Section gives a brief description of how to use the available software. Sections 2.1 and 2.2 describe how to change the code to easily run on your own graphs. These Sections rely on the classes 'MainUndirected' and 'Config', which should be enough for most purposes. In case of more specialized needs, Section 2.3 shortly describes how to write your own test cases, giving only the necessary information about the implementation.

2.1 Setting up the Configuration and Files

The class 'Config' is basically just a list of necessary configuration parameters. These can be roughly separated into two categories: input files and diagram plotting. The later includes the path to *gnuplot.exe* and the corresponding environment variable. Both need to be set to whatever this is on your computer. By default they are set to the standard path and variable in Ubuntu 10.4. Additionally it is possible to change the gnuplot terminal (default: *postscript eps color enhanced*) and the image format (default: *.eps*).

Concerning the input files for graphs only the 'Edges-Only-Format' for graphs is currently supported (see Section 3.1 on how to implement other readers). The files should contain two to five columns, the first two corresponding to the incident nodes of an edge. Nodes can be represented by any kind of label, string or numerical. In case of three or five columns the third column is interpreted as edge weight. If there are more than three columns, the last two give the node weights corresponding

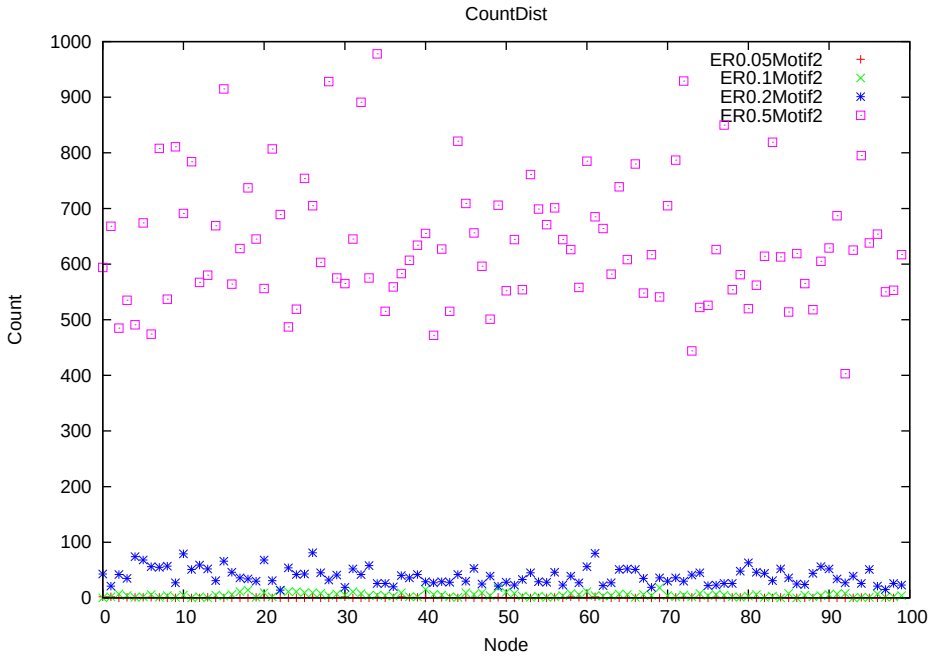

Figure 3: Motif 2 Counts per node of Erdös-Renyi graphs

to the incident nodes. The number of columns that should be used is defined in 'Config' as well as the deliminator of the individual columns (default: ' '). Additionally one can define if edge weights are integer or double. Note that while double can also be used in case of integer weights, using integers requires less space.

2.2 Configure the Test Cases

Having set up the parameters in 'Config' and created the graph files, this is merely an issue of selecting the appropriate test case and replacing the static variables *files* and *titles* in 'MainUndirected' with the paths to your graphs and their names. For selecting the appropriate test, it is best to read the comments to each test case, running them using the included 4 ER graphs to see if the results are as desired (this might be faster than using your own, probably larger, graphs). Note that in case you want to combine two of the tests for a large graph or a reasonable big number of graphs, it is better to write a combined one than using multiple tests sequentially.

2.3 Write own Tests

The typical flow of a test case is:

1. initiate the graph
2. choose a MotifAnalyzer *ma* and create one instance
3. run $ma.analyzeMotifs$ with desired parameters to create the output
4. run a one of the plotting functions given in 'Plot.java'

Creating a graph is done either by

$$\text{Graph g = new UndirectedNLGraph(new EdgesOnlyReader(}FILE\text{));}$$

or

$$\text{Graph g = new UndirectedAMGraph(new EdgesOnlyReader(}FILE\text{));}$$

where *FILE* is the path to the file describing the graph. In the first case the graph is represented using adjacency lists, in the second case using an adjacency matrix. The choice depends on the density and size of the graph. While both representations give the same results, the required time for calculation might differ.

To write your own test cases, you need to select the appropriate MotifAnalyzer. There are four abstract classes in the package motifs:

- MotifAnalyzer: offers the possibility to determine the singles described in Section 1 as well as weight distributions over all motif instances.

- MotifRetriever: extends MotifAnalyzer and additionally offers the option of writing instances to a file
- MotifNodeDistAnalyzer: extends MotifRetriever and additionally offers the option of creating the remaining distributions introduced in Section 1
- MotifNodeRetriever: extends MotifRetriever and is used for computing motifs locally, i.e., only instances with a specific node

The Analyzers in package 'motifs.undirected' all extend one of the above. See the respective comments to choose one.

After creating an object of type MotifAnalyzer, you can run it by using the function

analyzeMotifs(boolean edgeDist, boolean nodeDist, String result, Graph g),

where *edgeDist*, respectively *nodeDist*,is a flag for the computation of weight distributions. *result* is a prefix for the resulting files, i.e., for each generated file a postfix will be added. A list and description of these postfixes is given in the appendix.

This now generates the desired data. To plot it one needs to add the desired plot functions. These mostly take for input:

- the common path to the result path
- a list of titles of files that should be plotted
- the title for the plot
- a flag if the plot should be logarithmic

For example, the code in Listing produces the files 'graph1Counts.txt' and 'graph2Counts.txt' and a plot called Count comparing their motif signature using all 4-node undirected motifs:

```
String files = {graph1.graph, graph2.graph};
String titles = {graph1, graph2};
MotifAnalyzer mo = new FourUndirectedMotifAnalyzer(false, false);
for (int i = 0; i < files.length; i++) {
        try {
                Graph g = new UndirectedNLGraph(new EdgesOnlyReader(folder+ files[i]));
                mo.analyzeMotifs(false, false, ""+ titles[i], g);
               } catch (IOException e) {
                e.printStackTrace();
               }
}
Plot.plotCounts("", titles, "Count", false);
```

Listing 1: Example: All 4-Node motifs for two graphs

3 Background

3.1 Graph Structure

Directed and undirected graphs can be generated. One can use adjacency lists as well as adjacency matrices for storing neighborhood information. The first are more suitable for sparse graphs, the later for dense graphs. To optimize the space complexity, the user can set the configuration parameters to either use no edge weights, integer edge weights or double edge weights.

The type Graph basically just consists of flags for the above properties and an array of abstract type Node. Nodes have an index (equal to the position in the array), a label, a weight (default: 0) and neighborhood information. Up to now there are four different dynamic types for Node:

- NeighborListNode: Neighbors are given has a list of type Edge, where an instance can be either just the index of a Node, or index and edge weight.
- AdMatrixNodeBoolean: A boolean array the size of the graph, where a positive entry corresponds to an edge to the node with the respective index.
- AdMatrixNodeInteger: An integer array the size of the graph, where a positive entry corresponds to an edge with the given weight to the node with the respective index.
- AdMatrixNodeDouble: A double array the size of the graph, where a positive entry corresponds to an edge with the given weight to the node with the respective index.

For reading graphs from files there is the abstract type GraphReader. One needs to give the file name, if the graph is directed and the desired representation. The Graph Reader then create the graph as desired. Currently only Edges-Only-Format as described in Section 2.1 is supported. There is a GraphWriter, creating files from Graph objects. It is also possible to create a Graph by constructing an array with instances of Type Node and giving it to the constructor.

3.2 MotifAnalyzer

The class MotifAnalyzer offers the following functions:

public MotifAnalyzer(int motifsNr, int[] edges, int[] nodes, boolean edgeWeights, boolean nodeWeights)

Constructor: initializes the necessary arrays of length motifsNr: at least an array of type long for the motif counts; possibly also edgeweight and/or nodeweight

- motifNr: the number of motifs that are analyzed
- edges: number of edges in each motif
- nodes: number of nodes in each motif
- edgeWeights: use of edge weights
- nodeWeights: use of node weights

public void analyzeMotifs(boolean edgedist, boolean nodedist, String prefix, Graph g)

Method calling the function analyze(Graph g) and the functions for writing the output files; optionally the arrays for edge weight and node weight distributions are initialized

- edgedist: calculate edge weight distribution (only possible if there are edge weights!)
- nodedist: calculate node weight distribution (only possible if there are node weights!)
- prefix: prefix for the filenames, basically path to the resulting files
- g: graph that is analyzed

public abstract void analyze(Graph g)

abstract method for the actual analysis

- g: graph that is analyzed

protected void evaluateMotif(int index, Node[] motifNodes, Edge[] motifEdges)

evaluate a discovered motif: add one to the counts, optionally add weights and update distributions
is called from the method analyze(Graph g)
extended in subclasses for evaluating more properties

- index: index of motif in arrays count, edgeweight, nodeweight
- motifNodes: nodes in the motif
- motifEdges: edges in the motif

private void writeCountFile(String prefix)

Write the file prefix+Counts.txt (see Appendix), calculate remaining columns

- prefix: path to file

private void writeEdgeWFile(String prefix)

Write the file prefix+EdgeWeight.txt (see Appendix), calculate remaining columns

- prefix: path to file

private void writeNodeWFile(String prefix)

Write the file prefix+NodeWeight.txt (see Appendix), calculate remaining columns

- prefix: path to file

private void writeWeightDistFile(String prefix)

Write the files for the Weight Distribution over Motifs analysis (see Appendix), calculate remaining columns

- prefix: path to file

3.3 Plotting

Various functions for 2D-plotting are offered and described in 'Plot'. If the desired function is not present, one can use the class 'GnuplotWriter', which can create and execute a file for plotting. After creating an instance by either giving a file name or a FileWriter, one should first use the function 'writeHeader' which sets the properties of the plot.

public void writeHeader(String terminal, boolean lines, boolean logy, boolean logx, String xlabel, String ylabel, double[] yrange)

- terminal: terminal for the plot, e.g. png
- lines: define line styles
- logy: use logarithmic scale on y axis
- logx: use logarithmic scale on x axis
- xlabel: label for x axis
- ylabel: label for y axis
- yrange: range of y axis ([min,max])

Afterwards one calls the function 'writeOutput'for each desired plot

public void writeOutput(String output, String plotTitle, String[] files, String[] titles, int[] columns, String type, boolean style)

- output: name of output file
- plotTitle: title of plot
- files: paths to files that should be plotted
- titles: titles of files that should be plotted
- columns: columns that should be plotted in files (assumption: it is always first column as x!!)
- type: type of the plot, e.g. lines, points,...
- style: using the line styles defined in header (only possible if they were defined!!)

After the last call to 'writeOutput()' the writer has to be closed. Then it is possible to execute the file. One can either use the predefined gnuplot path and environment in 'Config' (see Section 2.1) or give them as parameters. in any case the path to the recently created file with the gnuplot code needs to be provided.

Appendix

Title	Columns	Implemented in Typ
Counts.txt	MotifNr, Counts, Counts Percentage	MotifAnalyzer
EdgeWeight.txt	MotifNr, Weight, Weight/Motif, Weight/Edge, Percentage	MotifAnalyzer
NodeWeight.txt	MotifNr, Weight, Weight/Motif, Weight/Edge, Percentage	MotifAnalyzer
EdgeDistMotif+*MotifNr*+.txt	Weight, Count, Fraction, CDF	MotifAnalyzer
NodeDistMotif+*MotifNr*+.txt	Weight, Count, Fraction, CDF	MotifAnalyzer
IndexCount.txt	Count, one column per motif	MotifNodeDistAnalyzer
IndexCountPerc.txt *	Count, one column per motif	MotifNodeDistAnalyzer
IndexEdgeWeight.txt	Weight, one column per motif	MotifNodeDistAnalyzer
IndexEdgeWeightPerc.txt *	Weight, one column per motif	MotifNodeDistAnalyzer
IndexNodeWeight.txt	Weight, one column per motif	MotifNodeDistAnalyzer
IndexNodeWeightPerc.txt *	Weight, one column per motif	MotifNodeDistAnalyzer
RetrieveMotif + *MotifNr*+.txt	list of nodes in motif, edgeweight, nodeweight of motif	MotifRetriever

* here the percentage is taken over all motifs, i.e., considering all Motifs with a given Count/Weight, this gives the fraction of them that are Motif 1,2,...