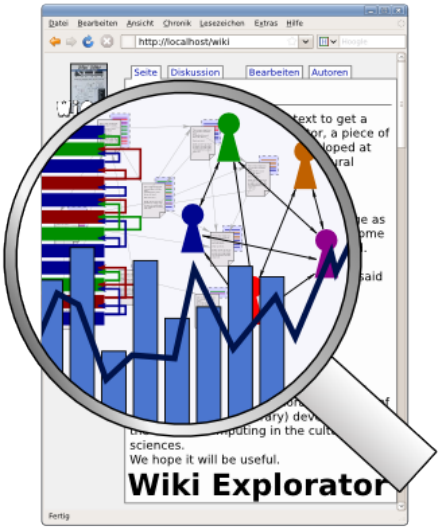
WikiExplorator Beginners Tutorial

Klaus Stein

September 7, 2009



Contents

1 Introduction

WikiExplorator is a ruby toolset and library for wiki exploration. It allows you to interactively analyse your wiki as well as to generate nice HTML or PDF reports customized to your needs. This includes various statistics as well as graph visualizations. The current version focusses on Mediawiki, other wiki engines can be used with some tricks.[1]

In this document I want to give some hints how to use WikiExplorator in a way that fits your needs. It is a tutorial, not a manual, this means that all steps given are examples, you will learn one or two ways to use a certain method or class, but you will not learn about every possible set of parameters and every method availible. So feel encuraged to look up the classes and methods used in this tutorial in the manual[2] and experiment with different parameter sets.

2 Installation

I unpacked the archive to the directory `mwparser` and installed Ruby[3], Graphviz, Gnuplot, R, LaTeX and some Ruby Gems and R packages without problems on (Debian) Linux. For detailed installation instructions see the `INSTALL` file.

3 First Steps

3.1 Getting Started

Enter the `mwparser` directory and make a copy of `mywikis.rb`. This file will be the personalized startup and project file, so all customation can be done inside this file. I recommend to copy this file and save it under another name as otherwise it would be overwritten when installing new versions.

```
~/mwparser/> ls
html  mediawiki  mediawiki.rb  mywikis.rb  Rakefile.rb  test  util
~/mwparser/> cp mywikis.rb wio.rb
~/mwparser/>
```
1

I name the new file `wio.rb` as our project and its wiki are named WiO. Now I open `wio.rb` with an editor and change it to fit my environment, looking for the following lines:

```
  def Mediawiki.mywiki(pw, options={})
    Wiki.open("wikidb", "localhost", "wikiuser", pw,
              {:language => 'de', :name => 'MyWiki'}.merge(options))
  end
```
2

[1] If someone wants to contribute a full featured module for some other wiki engine do not hesitate to contact me, I may be able to give useful hints.

[2] `http://wiki-explorator.rubyforge.org/doc/`

[3] version 1.8.7, version 1.8.6 should also work, if not send me a bug report. I testet WikiExplorator on Ruby1.9, but am not sure everything works as expected.

Using the class and method documentation[2] helps to understand the parameters. The relevant entry is `Mediawiki::Wiki.open`, where some of the parameters are described and a reference to `Mediawiki::DB.new` is given where the rest of the parameters is documented. To fit everything to my environment I change the name of the database to `wiodb`, give the database host, the database user and the name and language of the wiki[4].[5]

```
def Mediawiki.mywiki(pw, options={})
  Wiki.open("wiodb", "wiowiki.kinf.wiai.uni-bamberg.de", "wiouser", pw,
            {:language => 'de', :name => 'WiO'}.merge(options))
end
```
3

All of this information can be found by looking into `LocalSettings.php`[6].

So, let's see if everything works:[7]

```
~/mwparser/> ruby wio.rb
Tools for Social Network Analysis
[...]

Connecting to your wiki ...
Password: secret

Connecting to your wiki ...
Password:
connecting to database wiowiki.kinf.wiai.uni-bamberg.de/wiodb
connected.
Table wio_genres in DB not found
Table wio_roles in DB not found
users: 15
revisions: 1362
timeline length: 1352
firsttime: Di Mär 20 18:23:15 UTC 2007
lasttime: Di Jul 28 13:53:09 UTC 2009
Done.
Avg. # of users: 1.8348623853211
# of pages with more than one user: 57/109 (52.29%)

give "report" as command line parameter to create a pdf report of your wiki.

~/mwparser/>
```
4

This looks fine. Our wiki has 15 users and 1352 edits, had the first edit in March 2007 and the last one in July 2009.

[4] if our tables have some prefix "xxx", we would add `:prefix => 'xxx'` within the braces.
[5] if you have to use ODBC to access your database the second example should fit your needs
[6] note that `localhost` must be replaced by the actual hostname if you want to remotely access the database. In this case additionally Mysql must allow remote access (globally and for this user).
[7] the password can also be found in `Localsettings.php`

3.2 Interactive Usage

So let's see what WikiExplorator can tell about our wiki. We use the interactive ruby shell `irb`:[8]

```
~/mwparser/> irb -r wio
Tools for Social Network Analysis
...
irb(main):001:0>
```
5

The parameter "`-r wio`" means: require the library `wio` which is our project file.

The next step is to open our wiki:

```
irb(main):001:0> wiki = Mediawiki.mywiki('secret')
connecting to database wiowiki.kinf.wiai.uni-bamberg.de/wiodb
connected.
Table wio_genres in DB not found
Table wio_roles in DB not found
users: 15
revisions: 1362
timeline length: 1352
firsttime: Di Mär 20 18:23:15 UTC 2007
lasttime: Di Jul 28 13:53:09 UTC 2009
Done.
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
271 pages, 1362 revisions, 15 users>
irb(main):002:0>
```
6

Now the variable `wiki` holds the wiki object (see `Mediawiki::Wiki` in the manual) and we can start to investigate:

```
irb(main):002:0> wiki.users.length
=> 15
irb(main):003:0> wiki.pages.length
=> 109
irb(main):004:0> wiki.revisions.length
=> 1125
```
7

We ask the wiki to give us the users and count their length, we have 15 users, fine. Same for the pages, we have 109 pages. And finally for the revisions: 1125. But wait! Shouldn't this be 1352? There is something wrong.

This is caused by the fact that by default only namespace 0 (the main wiki namespace) is taken into account, so we habe 109 pages and 1125 revisions in namespace 0. We will learn soon how to change this by using filters (section 6).

For the moment we will see what the wiki can tell us using some ruby. I cannot provide a full-featured Ruby introduction within this tutorial but I will try to give some hints.[9]

[8] instead of `wio` you certainly use whatever you called your project file

[9] use `http://www.ruby-lang.org/` as a starting point for information about Ruby. You may at least need some Ruby knowledge when things start to get more sophisticated.

We may also be interested in a single user. Let's fetch the first one from our wiki's user collection: `wiki.users` gives a collection of User objects (see `Mediawiki::User`), and the method `first` gives the first entry in this collection:

```
irb(main):005:0> user = wiki.users.first
=> #<Mediawiki::User id=5 name="Regina">
```
8

In Ruby every method has a return value and irb indicates the returned object with "=>". Normally a condensed string representation of the object is given.

We could also have asked the wiki for user Regina:

```
irb(main):217:0> user = wiki.user_by_name('Regina')
=> #<Mediawiki::User id=5 name="Regina">
```
9

This certainly gives us the same object.

Now we can ask this user a lot of neat things:

```
irb(main):006:0> user.uid
=> 5
irb(main):007:0> user.name
=> "Regina"
irb(main):008:0> user.real_name
=> "Regina Meister"
irb(main):009:0> user.revisions.length
=> 105
irb(main):010:0> user.pages.length
=> 18
```
10

So Regina has 105 edits on 18 pages. Wouldn't it be nice to have a list of all users with their pages and edits? We only need to ask each user for his pages and his revisions, count them and collect this information:

```
irb(main):011:0> wiki.users.collect { |u|
                     [u.name, u.pages.length, u.revisions.length] }
=> [["Regina", 18, 105], ["Sissi", 0, 0], ["system", 1, 1], ["August", 1, 1],
    ["Tom", 17, 63], ["WikiSysop", 0, 0], ["Fritz", 20, 87], ["Olga", 21, 103],
    ["Klaus", 45, 151], ["Carla", 0, 0], ["Sam", 2, 9], ["Caro", 0, 0],
    ["Dan", 2, 6], ["Steffen", 71, 594], ["Helga", 2, 5]]
```
11

Some words about the syntax: we ask the wiki object about its users (call the method `users` on it), this returns a collection of users on which we now call the method `collect` which takes as a parameter. A block is a piece of code given in braces[10] taking one or more parameters given in pipes (`|u|`). `collect` calls the block given for every member of the collection and returns the results of the block in a large array.

Fortunately there is an auxiliary method which pretty prints (pp) these statistics (the columns are explained in the manual: `Mediawiki::Wiki.pp_userstats`):

[10] you can also use `do` ... `end` instead of `{` ... `}`

```
irb(main):013:0> wiki.pp_userstats
user         realname                 uid    e    p    e/p   se   fe   ke   ie
=============================================================================
August       August Kaiser              6    1    1   1.00    0    1    0    0
Carla        Carla Schach               8    0    0    nan    0    0    0    0
Caro         Carol Heart                9    0    0    nan    0    0    0    0
Dan          Dan Bosco                 15    6    2   3.00    2    4    0    1
Fritz        Fritz Lost                 7   87   20   4.35   55   27    0    1
Helga        Helga Mayer               10    5    2   2.50    3    1    0    0
Klaus        Klaus Stein                2  151   45   3.36   81   42    0    7
Olga         Olga Kranz                13  103   21   4.90   72   18    0    0
Regina       Regina Meister             5  105   18   5.83   73   31    0    0
Sam          Sam Hawkins               14    9    2   4.50    6    3    0    2
Sissi        Sissi Helfer              11    0    0    nan    0    0    0    0
Steffen      Steffen Blaschke           4  594   71   8.37  458   83    0   27
Tom          Tom Toppler               12   63   17   3.71   43   13    0    0
WikiSysop                               1    0    0    nan    0    0    0    0
system       System User                0    1    1   1.00    0    0    0    0
=> nil
```
12

Hm, and the other way around? How many users on each page?

```
irb(main):031:0> wiki.pages.collect { |p| p.users.length }
=> [1, 1, 1, 3, 1, 1, 7, 2, 1, 1, 1, 1, 3, 1, 1, 2, 1, 2, 1, 1, 1, 3, 2, 4, 2,
    5, 1, 2, 3, 1, 4, 1, 2, 3, 1, 2, 1, 2, 2, 2, 1, 3, 2, 1, 1, 2, 2, 4, 2, 2,
    2, 4, 1, 5, 1, 2, 2, 1, 1, 1, 2, 2, 1, 1, 1, 2, 1, 3, 2, 2, 1, 2, 1, 2, 1,
    2, 1, 2, 1, 3, 1, 2, 1, 2, 2, 4, 2, 3, 1, 2, 2, 2, 1, 1, 2, 2, 1, 4, 1, 1,
    1, 1, 4, 1, 1, 1, 1, 1, 2]
```
13

Not very clear. Perhaps a histogram with the number of pages for each user?

```
irb(main):037:0> wiki.pages.collect { |p| p.users.length }.stat_histogram
=> {5=>2, 1=>52, 7=>1, 2=>38, 3=>9, 4=>7}
```
14

That's better, but not good. We try some pseudographics:

```
irb(main):047:0> pu = wiki.pages.collect { |p| p.users.length }.stat_histogram
=> {5=>2, 1=>52, 7=>1, 2=>38, 3=>9, 4=>7}
irb(main):048:0> pu.keys.min.upto(pu.keys.max) { |k|
                     puts(('%2i: ' % k) + ('#' * pu[k])) }
 1: ##################################################
 2: ####################################
 3: #########
 4: #######
 5: ##
 6:
 7: #
=> 1
```
15

The `'%2i: '` is a printf expression as known from C.

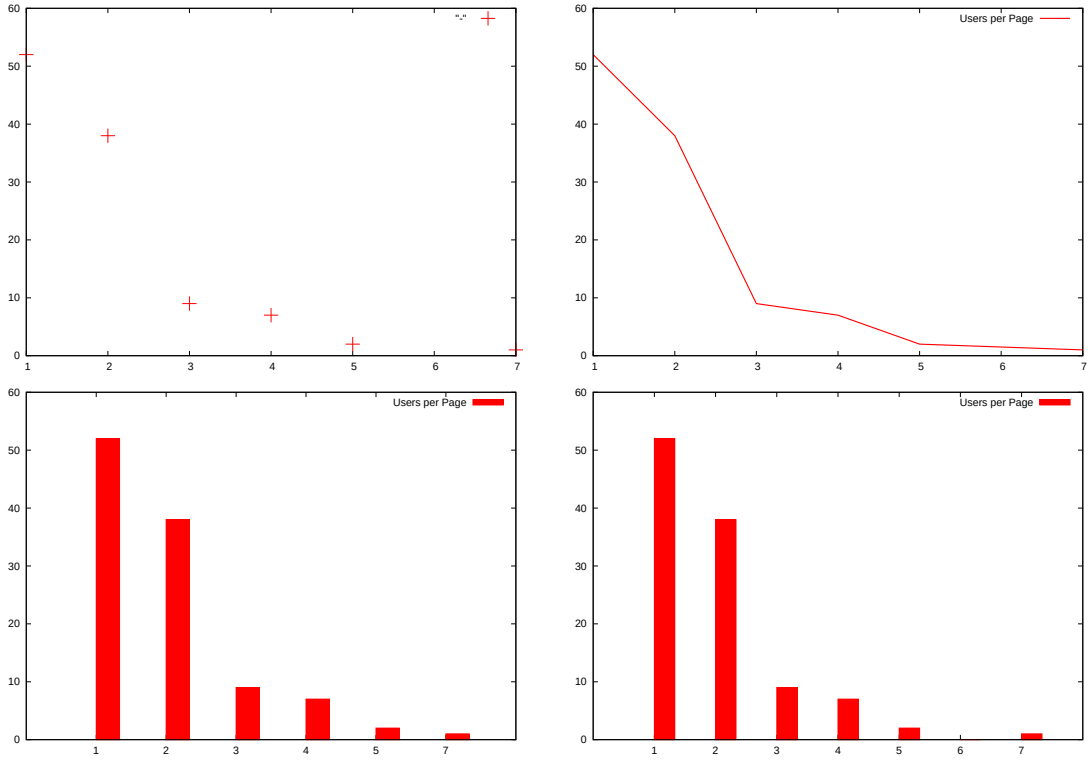


Figure 1: Using Gnuplot for data presentation

4 Gnuplot

Let's use gnuplot for a graphical representation of this histogram:[11]

```
irb(main):051:0> pu.gp_plot
=> #<Gnuplot:0xf6733cf4 ... >
irb(main):061:0> pu.sort.gp_plot(:title => 'Users per Page', :with => 'lines')
=> #<Gnuplot:0xf66f32a8 ...>
irb(main):070:0> pu.sort.gp_plot(:title => 'Users per Page',
                     :using => '2:xtic(1)', :with => 'histograms fill solid')
=> #<Gnuplot:0xf66b60c4 ...>
```

16

First we simply plotted the data using gnuplot (figure 1 top left), then the same using a line and adding a title (top right). Note that we had to sort the data before. Try out what happens otherwise. And finally we tried the histogram (bottom left). It looks fine but has one problem: it would be nice if we get an empty column with label 6 between 5 and 7. Let's see how to do this (figure 1 bottom right):

[11] using pu as defined in session 15

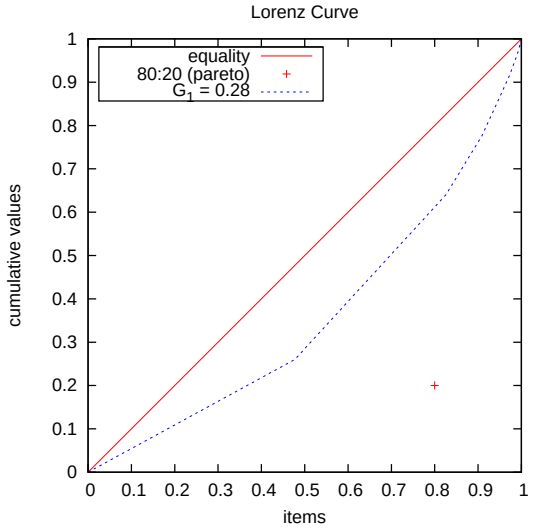


Figure 2: Lorenz curve for users per page

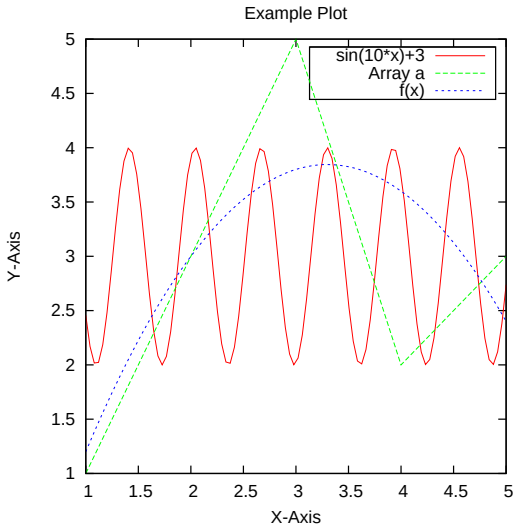


Figure 3: Extended gnuplot example

```
irb(main):072:0> (pu.keys.min..pu.keys.max).collect { |k| [k, pu[k]]
                     }.gp_plot(:title => 'Users per Page',
                     :using => '2:xtic(1)', :with => 'histograms fill solid')
=> #<Gnuplot:0xf66807d0 ...>
```
17

On the other hand, the data (users per page) is as if predestined to be presented as a Lorenz curve[12], so let's see what we get (figure 2):

```
irb(main):107:0> wiki.pages.collect { |p| p.users.length }.gp_plot_lorenz
=> #<Gnuplot:0xf64c3258 ...
 ... lines and lines of output ...
 ... >
```
18

Nice picture. But irb disturbed me with printing so many lines of output for the gnuplot object. We can fix this by adding another command with less output (we just use nil):

```
irb(main):107:0> wiki.pages.collect { |p| p.users.length }.gp_plot_lorenz; nil
=> nil
```
19

Fine. But gnuplot is more. See `http://gnuplot.sourceforge.net/demo_4.2/` for examples about what gnuplot can do. And certainly all of this is accessible from WikiExplorator, including function fitting etc. (see also the examples in `Gnuplot.new`, `Gnuplot.fit`). I give you a small example for extended gnuplot usage at figure 3 using an array `a` and the function $\sin(10x) + 3$:

[12] `http://en.wikipedia.org/wiki/Lorenz_curve`

```
irb(main):111:0> a = [[1,1], [2,3], [3,5], [4,2], [5,3]]
=> [[1, 1], [2, 3], [3, 5], [4, 2], [5, 3]]
irb(main):127:0> Gnuplot.new do |gp|
irb(main):128:1*   gp.add('sin(10*x)+3')
irb(main):129:1>   gp.add(a, :with => 'lines', :title => 'Array a')
irb(main):130:1>   gp.fit(:function => 'f(x) = a*x**2 + b*x + c', :via => 'a,b,c')
irb(main):131:1>   gp.title = 'Example Plot'
irb(main):132:1>   gp.set('key','box')
irb(main):133:1>   gp.set('xlabel', 'X-Axis', true)
irb(main):134:1>   gp.set('ylabel', 'Y-Axis', true)
irb(main):135:1>   gp.plot
irb(main):136:1> end
...
=> #<Gnuplot:0xf7a5e3c4 ... >
```
20

So one thing is left: we want the graphics to be saved, either as bitmap or as vector graphics. So let's plot our example array `a` to file[13]:

```
irb(main):184:0> a.gp_plot(:with => 'lines', :png => 'example.png')
=> #<Gnuplot:0xf6753400 ...>
irb(main):185:0> a.gp_plot(:with => 'lines', :svg => 'example.svg')
=> #<Gnuplot:0xf674c920 ...>
irb(main):186:0> a.gp_plot(:with => 'lines', :pdf => 'example.pdf')
=> #<Gnuplot:0xf6745c74 ...>
```
21

[13] you may need to use `:pspdf` instead of `:pdf` if your gnuplot has no native PDF support

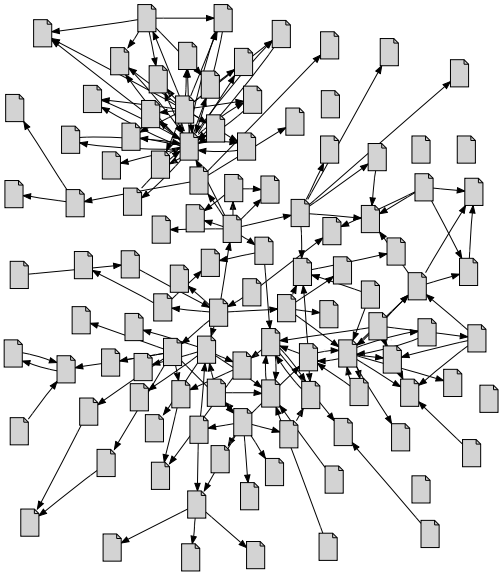


Figure 4: hyperlink network

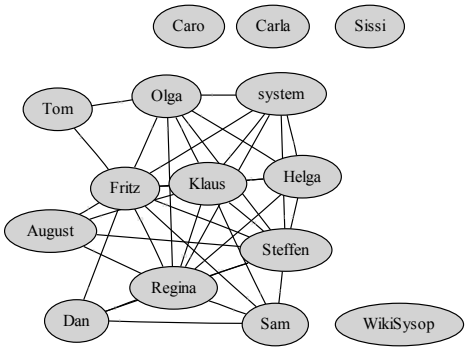


Figure 5: coauthorship network

5 Network Graphs

Many aspects of a wiki can be represented as networks. The most obvious networks are the page hyperlink graph (figure 4) where the nodes represent the wiki pages and the (directed) edges give the links between the pages, and the coauthorship network (figure 5) where the nodes represent the wiki authors and two authors are connected with a link if they edited the same page.[14]

As we will see soon a lot of varieties of these as well as different graphs are available.

5.1 First Steps

We start with asking the wiki for the coauthorshipgraph:

```
irb(main):030:0> cgraph = wiki.coauthorgraph
=> #<DotGraph:0xf779a2c8 ...>
```
22

This gives us a `DotGraph` object representing the network. This DotGraph object can now be used to compute various network measures and to visualize the graph in different ways. Most of the methods work on directed and undirected graphs, some also respect edge weights.

We start with some statistics about the degree distribution, sorted by user name (`DotGraph.pp_degrees`):

[14] more precisely: two authors are connected by an edge if there exists at least one page both did at least edit once.

```
irb(main):049:0> cgraph.pp_degrees(:sortby => :node, :up => true)
Node                            :  deg
August                          :    4
Carla                           :    0
Caro                            :    0
Dan                             :    4
Fritz                           :   10
Helga                           :    6
Klaus                           :    8
Olga                            :    7
Regina                          :    9
Sam                             :    5
Sissi                           :    0
Steffen                         :    9
Tom                             :    2
WikiSysop                       :    0
system                          :    6
=> nil
```
23

So Fritz is connected to most of the others, directly followed by Regina and Steffen. On a second thought, we are not interested on nodes not connected to others, so we remove them, and then we print our list again, now with user real names and sorted by degree:

```
irb(main):065:0> cgraph.remove_lonely_nodes; nil[15]
=> nil
irb(main):066:0> cgraph.pp_degrees(:sortby => :degree) { |u| u.real_name }
Node                            :  deg
Fritz Lost                      :   10
Steffen Blaschke                :    9
Regina Meister                  :    9
Klaus Stein                     :    8
Olga Kranz                      :    7
Helga Mayer                     :    6
System User                     :    6
Sam Hawkins                     :    5
August Kaiser                   :    4
Dan Bosco                       :    4
Tom Toppler                     :    2
=> nil
```
24

That's better. Now the ranking is obvious. We used the `pp_degrees` method in the example to get pretty output (`pp` stands for "pretty print"). For further processing we could have used `degrees` which gives the raw data. Some but not all methods have `pp_` counterparts for convenience. So when I use a `pp_` method in the following examples have a look at the pure method.

[15] You remember: the `nil` is only given to suppress the (rather lengthy) return output

5.2 R

If R and rsruby are installed and working (and the required R libraries are installed) we can access all of the network measures R (or rather the R library sna) provides using ruby. The method documentation states which DotGraph methods are based on R. So we can e. g. compute betweenness, closeness and others (for full R access see util/r.rb in the manual):

```
irb(main):093:0> cgraph.pp_betweenness(:sortby => :value)
Node                            :          betweenness
============================================================
Fritz                           :         8.8333333333
Steffen                         :         3.3333333333
Regina                          :         3.3333333333
Olga                            :         2.5000000000
Klaus                           :         1.7500000000
Sam                             :         0.2500000000
Helga                           :         0.0000000000
Dan                             :         0.0000000000
August                          :         0.0000000000
Tom                             :         0.0000000000
system                          :         0.0000000000
=> nil
irb(main):094:0> cgraph.pp_closeness(:sortby => :value)
Node                            :            closeness
============================================================
Fritz                           :         1.0000000000
Steffen                         :         0.9090909091
Regina                          :         0.9090909091
Klaus                           :         0.8333333333
Olga                            :         0.7692307692
Helga                           :         0.7142857143
system                          :         0.7142857143
Sam                             :         0.6666666667
August                          :         0.6250000000
Dan                             :         0.6250000000
Tom                             :         0.5555555556
=> nil
```

25

5.3 Graph Visualization

Network statistics are fine but we would rather be interested to see some nice graphics. WikiExplorator can create graph visualizations in different ways and with different layout algorithms using either graphviz or R.

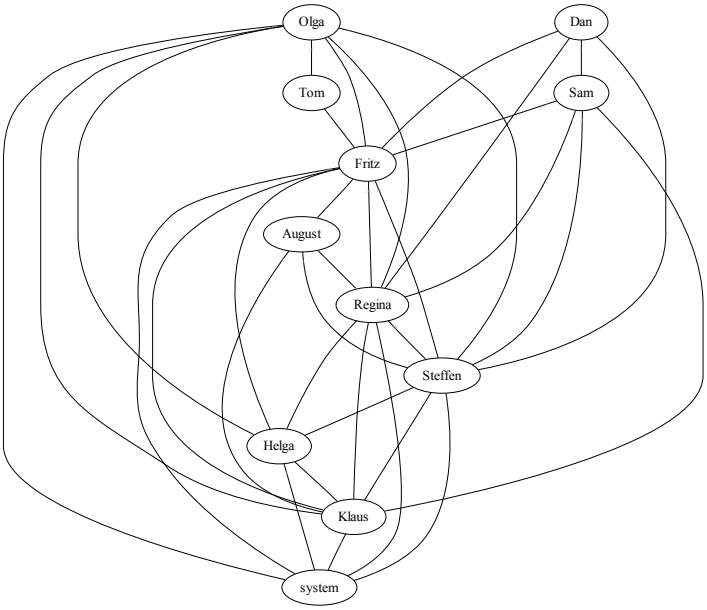


Figure 6: `dot` layout

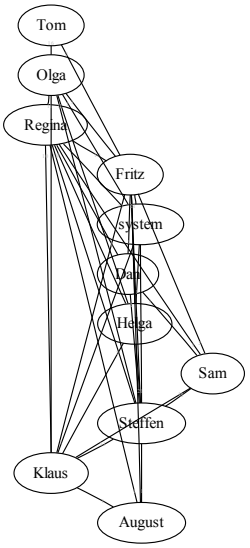


Figure 7: `fdp` layout

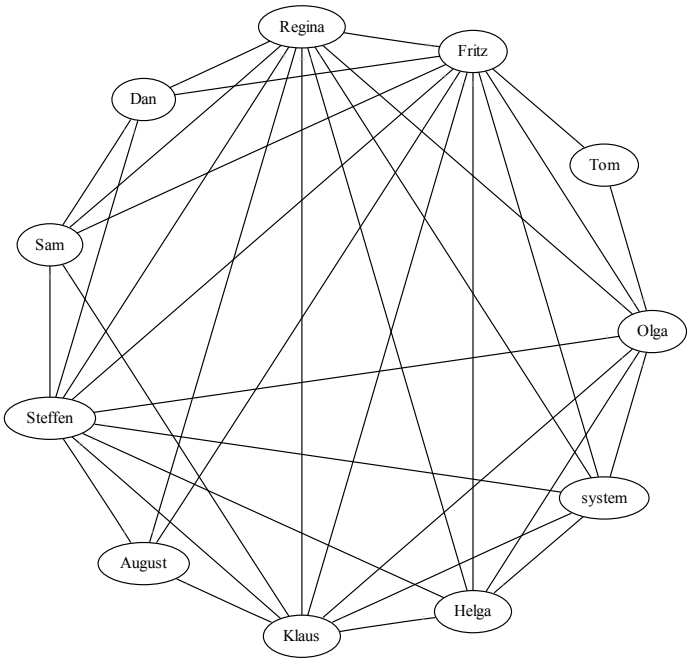


Figure 8: `circo` layout

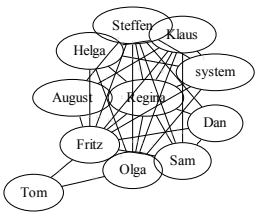


Figure 9: `twopi` layout

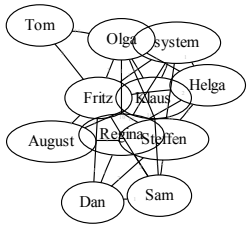


Figure 10: `neato` layout

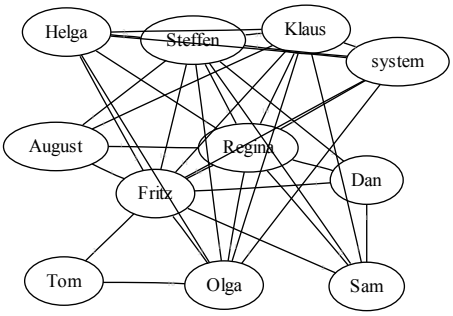


Figure 11: `twopi` without overlap

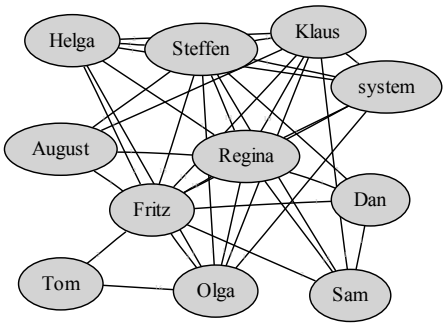


Figure 12: `twopi`: nodes in front of edges

5.3.1 Graphviz

graphviz can be used in two ways: by default the graphviz executables are called from ruby.[16] We start by using different graphviz layout engines[17] and create some PDFs:[18]

```
irb(main):107:0> cgraph.to_graphviz('gv_ca-dot.pdf','dot', :pdf)
=> true
irb(main):108:0> cgraph.to_graphviz('gv_ca-fdp.pdf','fdp', :pdf)
=> true
irb(main):109:0> cgraph.to_graphviz('gv_ca-circo.pdf','circo', :pdf)
=> true
irb(main):110:0> cgraph.to_graphviz('gv_ca-twopi.pdf','twopi', :pdf)
=> true
irb(main):111:0> cgraph.to_graphviz('gv_ca-neato.pdf','neato', :pdf)
=> true
```
26

This gives the graphs in figures 6 to 10.[19] These graphs obviously need some finetuning. For the `twopi` (fig. 9) and `neato` (fig. 10) layout the nodes overlap, and for `fdp`, `twopi` and `neato` (fig. 7–10) the edges go across the nodes. By passing additional parameters to graphviz we can solve this (see the graphviz documentation for available options):

```
irb(main):112:0> cgraph.to_graphviz('gv_ca-twopi-ol.pdf', 'twopi', :pdf,
                                    'overlap=false')
=> true
irb(main):113:0> cgraph.to_graphviz('gv_ca-twopi-olf.pdf', 'twopi', :pdf,
                                    'overlap=false', 'outputorder=edgesfirst',
                                    'node [style=filled]')
=> true
```
27

[16] If you have the ruby gv library installed you could require util/dotgraph-gv.rb for direct library access. This is not done by default because of a graphviz bug (reported and fixed since 2009-8-7).

[17] see the graphviz manpage and http://www.graphviz.org/ for details

[18] (you may need to use :pspdf instead, see footnote 13, Sec. 4), others (SVG, PNG, ...) are available.

[19] These graphs as well as the statistics before do not include the unconnected nodes visible in figure 5 as we removed them in session 24.

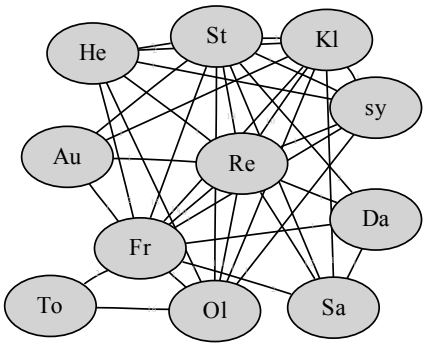


Figure 13: `twopi`: short labels

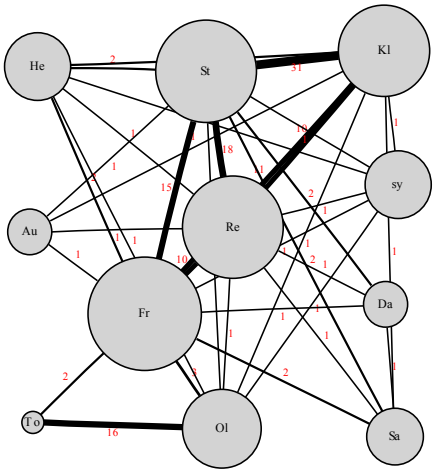


Figure 14: `twopi`: weighted sizes

The first call[20] gives figure 11, the second[21] figure 12. Try the same for `fdp` and `neato`! Currently the nodes have different size as the names (node labels) are of different length. Perhaps we should abbreviate all labels to two characters:

```
irb(main):115:0> cgraph.nodeblock { |node| node.name[0..1] }
=> #<Proc:0xf5ecf748@(irb):115>
irb(main):116:0> cgraph.to_graphviz('gv_ca-twopi-olf-short.pdf', 'twopi', :pdf,
                                    'overlap=false', 'outputorder=edgesfirst',
                                    'node [style=filled]')
=> true
```
28

In the first line we set the `nodeblock` of this graph. The nodeblock is called for each node to create its label and should either return a string (used as label) or an array (used as node parameters). See `DotGraph.nodeblock` for details.

I used this to create the hyperlink graph (figure 4):

```
irb(main):018:0> wiki.pagegraph {''}.to_graphviz('pagegraph.pdf', 'neato', :pdf,
                    'outputorder=edgesfirst', 'overlap=false',
                    'node [shape=note, style=filled, width=.34]')
=> #<IO:0xf765e454>
```
29

Here the nodeblock is given on creation and returns an empty string for any node. By adding some width and shape directives I got these nice paper icons as node shapes to represent our wiki pages.

And for completeness, here is figure 5 (we create a new coauthorgraph from the wiki, so here the unconnected nodes are not removed and therefore displayed in the graph):

[20] `'overlap=false'` tells graphviz to shift the nodes apart from each other

[21] `'outputorder=edgesfirst'` tells graphviz to first draw the edges, so the nodes are printed over the edges. This only has an effect if the nodes are solid, so we add `'node [style=filled]'`

```
irb(main):028:0> wiki.coauthorgraph.to_graphviz('gv_coauthorgraph-full.pdf',
                     'neato', :pdf, "outputorder=edgesfirst", "overlap=false",
                     "node [style=filled]")
=> #<IO:0xf7a7554c>
```
30

Finally, we want to compute node sizes dependent on node degrees and edge width dependent on link weights[22]:

```
irb(main):161:0> cgraph.nodeblock { |node|
                                    size = (cgraph.n_degree(node)/10.0).to_s;
                                    ["label=#{node.name[0..1]}",
                                     'width=' + size, 'height=' + size] }
=> #<Proc:0xf5c80dd4@(irb):161>
irb(main):173:0> cgraph.to_graphviz('gv_ca-twopi-degwidth.pdf', 'twopi', :pdf,
                     'overlap=false', 'outputorder=edgesfirst',
                     'node [fontsize=8, style=filled, fixedsize]') { |weight|
                        ["penwidth=#{weight**0.5}", "label=#{weight}",
                         'fontsize=7', 'fontcolor=red' ] }
Warning: node 'u6', graph 'G' size too small for label
Warning: node 'u12', graph 'G' size too small for label
Warning: node 'u15', graph 'G' size too small for label
=> #<IO:0xf5bb9950>
```
31

We first set the nodeblock. We use the degree of the node to compute the `size`: we ask `cgraph` for the degree of the node and divide it by 10.0.[23] The method `to_s` converts the result to string. Now we asseble an array with three entries: the node label set to the first two characters of the user name, the node width and the node height, both set to `size`.[24] And finally we ask `cgraph` to create the graph. You may notice that we appended an additional block to this call. This block is called for each edge with the edge weight as parameter, similar to nodeblock.

Graphviz allows much more than the few examples shown here,[25], if you find expressive and impressive visualizations feel free to send me examples (source code and graphics). The parameters are described at `http://www.graphviz.org/doc/info/attrs.html`, the graphviz homepage provides elaborate user manuals for all layout algorithms. Use `to_dotfile` to see the outcome of the parameters:

```
irb(main):218:0> cgraph.to_dotfile('cgraph.dot', 'outputorder=edgesfirst',
                     'overlap=false', 'node [style=filled]')  { |weight|
                           ["penwidth=#{weight**0.5}", "label=#{weight}"] }
=> #<File:~/mwparser/cgraph.dot (closed)>
```
32

The resulting file `cgraph.dot` can be opened with any text editor.

[22] `coauthorgraph` weights edges between authors with the number of pages they are coauthors on.
[23] the divisor has to be a Float, otherwise ruby would use Integer division
[24] You may notice usage of single (') and double (") quotes. This is similar to shells like sh or bash: strings in double quotes (") are subject to substitution, in our example this means that the result of the ruby code given in `#{...}` is inserted in the string.
[25] see the gallery at `http://www.graphviz.org/Gallery.php`

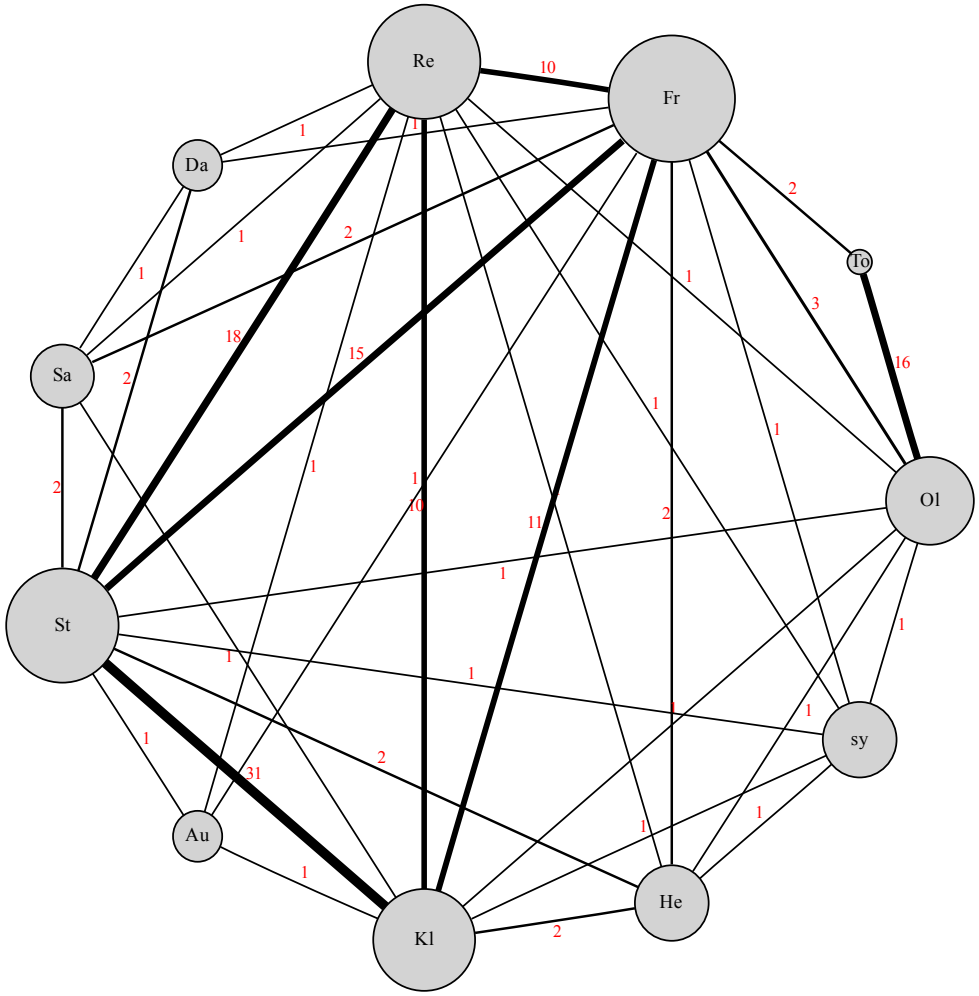


Figure 15: `circo`: weighted sizes

Lets close with one final example. We use (nearly) the same parameters as before and just change the layout engine (figure 15):

```
irb(main):180:0> cgraph.to_graphviz('gv_ca-circo-degwidth.pdf', 'circo', :pdf,
                    'overlap=false', 'outputorder=edgesfirst',
                    'node [fontsize=11, style=filled, fixedsize]')  { |weight|
                       ["penwidth=#{weight**0.5}", "label=#{weight}",
                        'fontsize=10', 'fontcolor=red' ] }
Warning: node 'u6', graph 'G' size too small for label
Warning: node 'u12', graph 'G' size too small for label
Warning: node 'u15', graph 'G' size too small for label
=> #<IO:0xf5b7fc00>
```

33

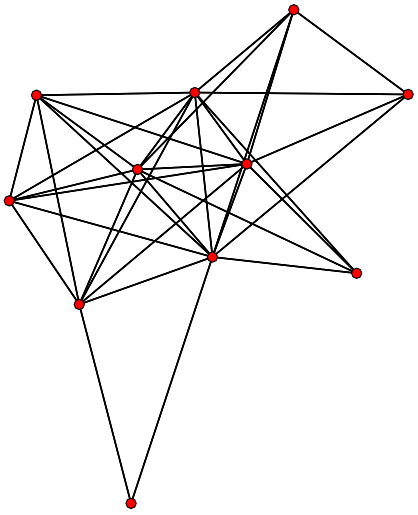


Figure 16: Plotting using R

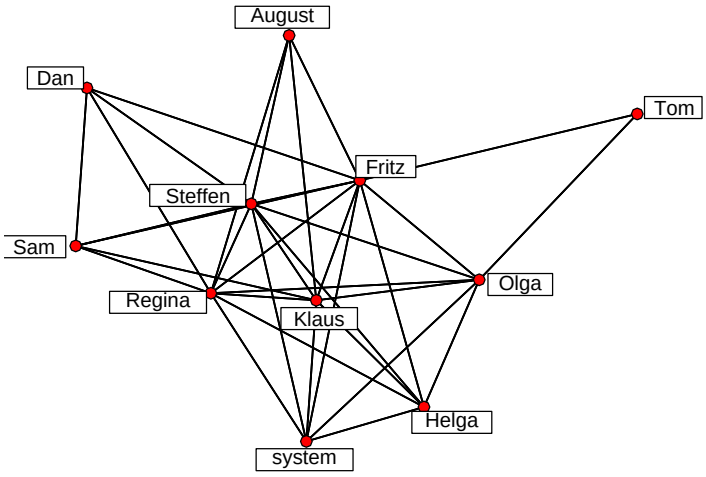


Figure 17: Plotting with labels using R

5.3.2 R

Graphviz allows for rather pretty and sophisticated graph prints, but has the drawback not to display graphs on screen (which was rather convenient with gnuplot). Thankfully we can use R for this (figure 16):

```
irb(main):182:0> dev = cgraph.r_plot
=> 2
irb(main):183:0> cgraph.r_plot_close(dev)
=> "null device"=>1
```
34

As you can see `cgraph.r_plot` returns a number which indicates the R plot device and which can be used to close the graph window. On some OS you may just close the window using its close button, on others this does not work, there you have to use `r_plot_close`.

Unfortunately changing attributes can get somehow cumbersome using R syntax, well, at least for me, if you know R from heart things may be different for you (figure 17):[26]

```
irb(main):190:0> cgraph.r_plot(:displaylabels => TRUE,
                     :label => lambda { |nw,r|
                        r.get_vertex_attribute(nw, "attr")}) { |u| u.name }
=> 2
irb(main):191:0> DotGraph.r_plot_close(2)
=> "null device"=>1
```
35

So I use `r_plot` for a fast interactive graph overview and stay with graphviz for prints.

[26] this example uses `DotGraph.r_plot_close` which works as well and ist independent of a certain graph object.

6 Using Filters

Enough of playing with graphs and networks, back to the wiki.

As stated before the wiki is not seen raw but through a view which is customized by a filter. Filters are the way to see cutouts of a wiki, including or excluding certain users, namespaces, and to go back in time, to see how the wiki looked like years ago or what happened within a certain timespan.

6.1 First steps

When asking the wiki for the number of pages by default I only get pages from the main namespace (0). But we certainly can change this:

```
irb(main):199:0> wiki.pages.length
=> 109
irb(main):200:0> wiki.filter.include_all_namespaces
=> #<Set: 0, 6, 1, 2, 8, 14, 3, 4, 10>
irb(main):201:0> wiki.pages.length
=> 271
```

36

We first ask the wiki for its pages. After that `wiki.filter` gives us the active filter of the wiki on which we call the method `include_all_namespaces`. As we can see from the return value this automagically collects all namespaces found in the wiki. And finally we ask the wiki again for its pages, and suddenly their number raises from 109 to 271.

We have two users in our wiki who are not part of our normal staff: “system” and “WikiSysop”.[27] Perhaps we should exclude them from our statistics:

```
irb(main):230:0> f = wiki.filter
=> #<Mediawiki::Filter:0xf7669e6c ...>
irb(main):231:0> wiki.revisions.length
=> 1362
irb(main):232:0> wiki.user_by_name('system').revisions.length
=> 2
irb(main):233:0> wiki.user_by_name('WikiSysop').revisions.length
=> 1
irb(main):234:0> f.deny_user('system', 'WikiSysop')
=> ["system", "WikiSysop"]
irb(main):235:0> wiki.revisions.length
=> 1359
```

37

We ask the wiki for its filter and save it in `f`. The wiki currently has 1362 revisions, the “system” user has two revisions and “WikiSysop” has one. Now we tell the filter to exclude these two users and the wiki shows us 1359 ($= 1362 - 2 - 1$) revisions, so this looks fine.

[27] While the “WikiSysop” is an ordinary wiki user with special rights, the “system” user is special. It does not show up in the user database and you cannot login as “system” user, but it is owner of all revisions done internally by Mediawiki (e. g. as result of mass uploads etc.).

How does this work? Well, when we ask the wiki for its users, we get a UsersView of the wiki which uses the given filter:

```
irb(main):240:0> users = wiki.users
=> #<Mediawiki::UsersView:0xf5d4dba4 ...>
irb(main):241:0> users.length
=> 13
irb(main):242:0> f.undeny_user('WikiSysop')
=> ["WikiSysop"]
irb(main):243:0> users.length
=> 14
```
38

What we see here is that changing the filter affects the corresponding view.

6.2 Multiple Filters

In the last section we changed the current filter of the wiki to get different views. This does not work if we want to deal with different views in parallel: we need more than one filter. So let's copy our current filter and see what we can do with it:

```
irb(main):244:0> f2 = f.clone
=> #<Mediawiki::Filter:0xf5d43618 ...>
irb(main):245:0> f2.namespace = 0
=> 0
irb(main):246:0> wiki.revisions(f2).length
=> 1124
irb(main):247:0> wiki.revisions.length
=> 1360
```
39

We use `clone` to get a real copy of the filter `f` and save it in `f2`,[28] after that we restrict the namespace of `f2` to 0 and then we use `f2` for the `revisions` view. Many Wiki methods[29] take a filter as optional parameter.[30]

A cloned filter inherits the attributes of the original one. Alternatively we can create a clean filter for our wiki:

```
irb(main):249:0> f3 = Mediawiki::Filter.new(wiki)
=> #<Mediawiki::Filter:0xf5cf2628 ...>
```
40

So `f3` has all attributes set to start values (namespace 0, no excluded users, ...).

See `Mediawiki::Filter` to learn about filter attributes.

6.3 Back and Forth in Time

One nice feature of a wiki is the page history. We can reconstruct how the wiki looked like months or years ago,[31] and we do this using filters.

[28] cloning is important, otherwise both varfiables would point to the *same* object!
[29] i. e. graph generation and statistics methods
[30] send a bug report if I missed a method.
[31] WikiExplorator currently does not handle deleted pages, this is considered a bug.

So let's see how the wiki looked like a year ago:

```
irb(main):260:0> f3.revision_timespan
=> Di Mär 20 18:23:15 UTC 2007..Di Jul 28 13:53:09 UTC 2009
irb(main):261:0> f3.endtime = '2008-8-1'
=> "2008-8-1"
irb(main):262:0> f3.revision_timespan
=> Di Mär 20 18:23:15 UTC 2007..Fr Aug 01 00:00:00 +0200 2008
irb(main):266:0> wiki.pages(f3).length
=> 84
irb(main):268:0> wiki.pages(f2).length
=> 109
```

41

Initially `f3` includes the whole timespan from March 20, 2007 to July 28, 2009. We set the endtime to August 1, 2008 and ask again for the timespan, fine, worked like expected.

Now we ask the wiki for the pages within this timespam (84) and compare it to the number of pages until now (109). So obviously we got 25 ($= 109 - 84$) new pages added in the last year.

Ok, how many revisions within the last four weeks?

```
irb(main):270:0> f3.full_timespan
=> Di Mär 20 18:23:15 UTC 2007..Di Jul 28 13:53:09 UTC 2009
irb(main):271:0> f3.starttime = f3.endtime - 60*60*24*7*4
=> Di Jun 30 13:53:09 UTC 2009
irb(main):272:0> f3.revision_timespan
=> Di Jun 30 13:53:09 UTC 2009..Di Jul 28 13:53:09 UTC 2009
irb(main):273:0> wiki.revisions(f3).length
=> 10
```

42

We first set `f3` to full timespan, set the new starttime by subtracting four weeks in seconds from the endtime, controlled that it worked and asked for the number of revisions. Not much traffic in our example wiki the last four weeks.

Wouldn't it be nice to have this for the whole timeline? That's easy as the wiki helps us generating a custom timeraster:

```
irb(main):270:0> f3.full_timespan
=> Di Mär 20 18:23:15 UTC 2007..Di Jul 28 13:53:09 UTC 2009
irb(main):275:0> tr = wiki.timeraster(:step => :week, :zero => :week)
=> [So Mär 18 00:00:00 +0100 2007, So Mär 25 00:00:00 +0100 2007, ...,
                                   ..., So Aug 02 01:00:00 +0200 2009]
irb(main):279:0>  r1 = tr.collect { |t| f3.endtime = t;
                                      [t, wiki.revisions(f3).length] }
=> [[So Mär 18 00:00:00 +0100 2007, 0], [So Mär 25 00:00:00 +0100 2007, 18], ...,
                                   ..., [So Aug 02 01:00:00 +0200 2009, 1125]]
irb(main):284:0> r1.gp_plot(:with => 'lines'); nil
=> nil
```

43

First we reset `f3` to full timespan (again). Then we ask the wiki to generate a weekly timeraster and use this timeraster to compile an array of arrays with the endtime as first and the number of revisions up to this time as second entry. And finally we use gnuplot

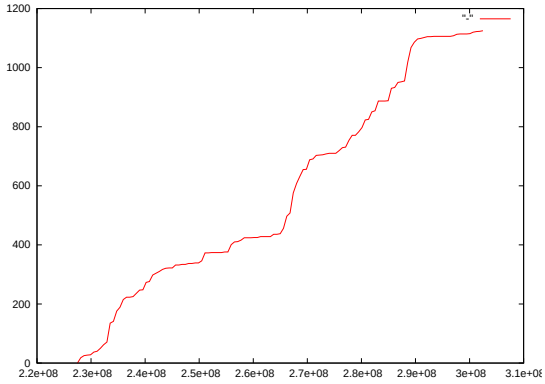


Figure 18: weekly revisions (cumulative)

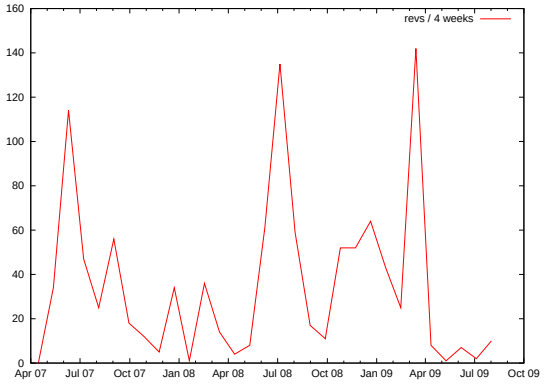


Figure 19: monthly revisions

to plot the revision growth in time (figure 18).

In the last example we only changed endtime, so the number of revisions grew. Now we want to see how the number of new revisions changes in time. The following example could be written slightly more simple when using the `enumerator` module available in ruby 1.8.7 and later (see `Mediawiki::Wiki.timeraster` for an example using enumerator). We use a time raster of four weeks:

```
irb(main):339:0> tr = wiki.timeraster(:step => 60*60*24*7*4, :zero => :week)
=> [So Mär 18 00:00:00 +0100 2007, So Apr 15 01:00:00 +0200 2007, ...]
irb(main):340:0> start = tr.shift
=> So Mär 18 00:00:00 +0100 2007
irb(main):341:0> r2 = tr.collect { |t| f3.revision_timespan = (start..t);
                                       start = t;
                                       [t, wiki.revisions(f3).length] }
=> [[So Apr 15 01:00:00 +0200 2007, 0], ... ]
irb(main):381:0> Gnuplot.new do |gp|
irb(main):382:1*   gp.set('xdata', 'time')
irb(main):383:1>   gp.set('timefmt', '%Y-%m-%d', true)
irb(main):384:1>   gp.set('format x', '%b %y', true)
irb(main):385:1>   gp.add(r2, :with => 'lines', :title => 'revs / 4 weeks',
                             :timefmt => '%Y-%m-%d')
irb(main):386:1>   gp.plot
irb(main):387:1> end
=> #<Gnuplot:0xf5d8a61c ...>
```

44

As you can see I put some efforts in gnuplot to get nice formatted output. So I hope you got an idea about what filters can do for you, even if sometimes the direct way is faster (this results in a graph very close to session 43 (figure 18)):

```
irb(main):416:0> c = 0; wiki.revisions.sort_by { |r| r.timestamp }.collect { |r|
                          [r.timestamp, c += 1] }.gp_plot(:with => 'lines')
```

45

Here we directly use the revision timespans to generate the graph.

7 Load and Save

Until now we played with a wiki we loaded directly from the database (we come back to this at the end of this section, see 7.2).

7.1 Loading and Dumping

Sometimes it would be nice to take the wiki with us to be able to do some statistics, reporting, working through this tutorial etc. on our laptop while sitting in the train without internet access. To support this we can dump the wiki on disk to load it later.

7.1.1 Remove Sensible Data

Taking a wiki with us means dumping wiki contents on our hard disk. You may want to analyze the wiki of your customer containing sensible data, and you may want to take it with you on your laptop, but your customer would not like to see sensible data leaving the intranet.

As we nearly do not need the contents of the wiki pages for most analyses the `obliterate` method is our friend. This method removes page contents, user names, page titles etc. from the wiki and keeps only the structure which is enough to do nearly anything we want (except for text corpus and text size analysis):

```
irb(main):471:0> page = wiki.page_by_id(1)
=> #<Mediawiki::Page id=1 title="Hauptseite">
irb(main):472:0> page.content
=> "'''Wikis in Organizations''' ..."
irb(main):473:0> wiki.obliterate
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
                                          271 pages, 1362 revisions, 15 users>
irb(main):474:0> page.content
=> ""
irb(main):475:0> page
=> #<Mediawiki::Page id=1 title="p1">
```
46

What exactly is removed/kept by `obliterate` is freely selectable by the user (see `Mediawiki::Wiki.obliterate` for details).

7.1.2 Marshal Dump and Load

Now we can dump our wiki to disk for later usage:

```
irb(main):478:0> wiki.marshal_save('wio.marshal')
=> #<File:~/mwparser/wio.marshal (closed)>
irb(main):479:0> exit
```
47

We leave irb, have a break, meet people and do real world stuff, you know, things that really matter, thinks you don't need a computer for, and later we come back and start

again, now loading our wiki from file:[32]

```
~/mwparser/> irb -r wio
...
irb(main):001:0> wiki = Mediawiki::Wiki.marshal_load('wio.marshal')
MARSHAL load...
done
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
                                        271 pages, 1362 revisions, 15 users>
```
48

I have obliterated marshal dumps of all of the wikis I am working on and use only these for daily work.[33]

7.1.3 YAML Dump and Load

Marshalling is fast and reliable, but it dumps the wiki to some binary format, so your customer cannot control what data you want to take with you. YAML is much slower than marshal but writes human readable files so you can show them what you want to take away:

```
irb(main):002:0> wiki.yaml_save('wio.yaml')
preprocessing (detach links)...
YAML save (may take some time)...
postprocessing (attach links)...
done
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
                                          271 pages, 1362 revisions, 15 users>
irb(main):004:0> exit
```
49

This time we do some sports before we come back. Oh, and have a look into "`wio.yaml`" to see what's inside (figure 20)

```
~/mwparser/> irb -r wio
...
irb(main):001:0> wiki = Mediawiki::Wiki.yaml_load('wio.yaml')
YAML load...
postprocessing (attach links)...
done
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
                                        271 pages, 1362 revisions, 15 users>
```
50

YAML is an ordinary ruby library, but as you can see by the messages above I had to provide a customized version as yaml has problems with large nested cyclic structures. So please only use the methods given above to YAML save and load your wiki and not the original ones privided by the YAML lib.

[32] "`-r wio`" is needed to pull in the WikiExplorator library. As we do not need a customized database connector we could also have used "`-r mediawiki/full`" instead.

[33] this has the nice side effect that it works on platforms where installing a Mysql connector gets laborious.

```
--- &id001 !ruby/object:Mediawiki::Wiki
dburl: wiowiki.kinf.wiai.uni-bamberg.de/wiodb
filter: !ruby/object:Mediawiki::Filter
  revision_timespan: !ruby/range
    begin: &id002 2007-03-20 19:23:15 +01:00
    end: &id003 2009-07-28 15:53:09 +02:00
    excl: false
  ...
language: de
name: WiO
...
pages_id:
  1440: !ruby/object:Mediawiki::Page
    len: 864
    namespace: 0
    pid: 1440
    ...
  1641: !ruby/object:Mediawiki::Page
    ...
  ...
pages_title:
  ...
revisions_id:
  1556: !ruby/object:Mediawiki::Revision
    page: 1425
    rid: 1556
    text: 1556
    timestamp: &id156 2007-05-23 13:26:52 +02:00
    user: 5
    ...
  2405: !ruby/object:Mediawiki::Revision
    ...
  ...
texts_id:
  1823: !ruby/object:Mediawiki::Text
    text: ""
    tid: 1823
    ...
  ...
users_id:
  5: !ruby/object:Mediawiki::User
    name: u5
    revisions:
      ...
    uid: 5
    ...
  0: !ruby/object:Mediawiki::User
    ...
  ...
users_name:
  ...
version: 1.8
```

Figure 20: YAML dump of the wiki (excerpt)

7.2 Wiki.open

At the beginning of this tutorial (section 3.1) we created our Wiki object by adapting some values in `mywikis.rb`. We could have done this directly:

```
~/mwparser/> irb -r mediawiki/full
...
irb(main):001:0> wiki = Mediawiki::Wiki.open("wiodb",
                        "wiowiki.kinf.wiai.uni-bamberg.de", "wiouser", 'secret',
                                                :language => 'de', :name => 'WiO')
connecting to database wiowiki.kinf.wiai.uni-bamberg.de/wiodb
...
Done.
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
                                          271 pages, 1362 revisions, 15 users>
```

[52]

Using a startup file is just convenient.

Please have a detailed look at `Mediawiki::Wiki.open` (and `Mediawiki::DB.new`) in the manual for the various parameters. It is important to set the right wiki name, the right language and give mappings for additional namespaces to get correct link structures.

Mediawiki unfortunately does not put everything into the database, the correct interpretation of various data depends on the values in `Localsettings.php`, so I had no choice than to give these as parameter.

Two of the parameters allow special customation: `:ips` and `:uid_aliases`.

7.2.1 :ips

If a wiki allows anonymous edits the IP address of the editing anonymous user is saved for each revision. Normally it is not useful to distinguish between different IPs as the same user may come back with a different IP the next day (at least if he has a dialup connection to the internet), so all anonymous edits are mapped to one default IP user.

Within an intranet things may different as the workstations of the staff may have fixed IP addresses. In this case simply add "`:ips => true`" to the parameters and a new user is created for each IP.

7.2.2 :uid_aliases

On many of the wiki we analyzed we found users with more than one login, either because login name policy of the company changed after the wiki started or users sometimes logged in and sometimes forgot and did anonymous edits from a fixed IP address etc.[34]

To deal with these situations WikiExplorator allows to merge users. For example, in our example wiki I know that "WikiSysop" and "Klaus" are the same person, so we may want to merge them[35]:

[34] we even had a company where we could map sets of IP addresses to certain roles but not one IP address to one user.

[35] on the other hand it may be a bad idea, but, well, we do it to have an example.

page_id	genres
1	portal
1410	description>project
1424	list>todo
1471	list>collection, article
1477	howto, article
1554	description>project
1623	calendar, event
	...

Figure 21: `wio_genres` table

user_id	roles
2	Support
4	Staff
5	ProjectManagement
6	ProjectManagement
7	Staff
8	Interns
10	Interns
	...

Figure 22: `wio_roles` table

```
irb(main):002:0> wiki.user_by_name('Klaus')
=> #<Mediawiki::User id=2 name="Klaus">
irb(main):003:0> wiki.user_by_name('WikiSysop')
=> #<Mediawiki::User id=1 name="WikiSysop">
irb(main):004:0> wiki = Mediawiki::Wiki.open("wiodb",
                    "wiowiki.kinf.wiai.uni-bamberg.de", "wiouser", 'secret',
                    :language => 'de', :name => 'WiO', :uid_aliases => {1 => 2})
connecting to database wiowiki.kinf.wiai.uni-bamberg.de/wiodb
connected.
Table wio_genres in DB not found
Table wio_roles in DB not found
...
Done.
=> #<Mediawiki::Wiki WiO: wiowiki.kinf.wiai.uni-bamberg.de/wiodb,
                                              271 pages, 1362 revisions, 14 users>
irb(main):017:0> wiki.user_by_id(1)
=> nil
```
53

It worked. We first looked up the ids of "Klaus" and "WikiSysop" and then opened the wiki aliasing 1 to 2. All revisions of user 1 now belong to user 2.

7.2.3 wio_roles and wio_genres

Every time we open the wiki it claims that it did not find the tables `wio_genres` and `wio_roles` in the database. These tables are an add-on by us and do normally not exist in a Mediawiki installation. We use them to map each page to its genres (this is *not* the same than its category) and each user to its roles. So if you know the role of every user in the company (manager, programmer, culprit, ...) you can create a `wio_roles` table and later you can use filters to get different statistics for different roles (to prove the managers do all the work or whatever).

`wio_genres (page_id, genres)` maps page ids to lists of genres seperated by ",". `wio_roles (user_id, roles)` maps user ids to lists of roles also separated by "," (see figures 21 and 22).

In figure 21 you see strings connected with ">". This is output from TAMS/GTAMS[36], the tool we used to do text analysis of the wiki pages. No splitting is done at ">", so "list>collection" is taken as one word. We can nevertheless filter all pages of genre "list" using a regular expression: `filter.genregexp = /^list\b/`.

So feel free to use these tables if they seem useful to you.[37]

7.2.4 wio_user

Accessing a wiki database remotely means that the database administrator has to provide a database user with read access for at least the tables "`user`", "`user_groups`", "`text`", "`page`" and "`revision`".

The user table contains various sensible data, especially user passwords. While WikiExplorator denies to read this fields it would be better if it had no access at all. This can be accomplished by creating a view called "`wio_user`" which exposes only some of the fields from the "`user`" table. If "`wio_user`" is found, "`user`" is not read at all.

Wiki farms often have one common user table for all wikis. This is a problem for Wiki Analysis as most users only belong to one or two wikis. "`wio_user`" can also help here: simply create this table for each of the wikis to be analyzed by copying the users which show up in at least one revision.[38]

7.3 Wiki.open_XML

Reading the database directly is fine if we can connect to it from our machine which is not always true. If we do not have database access at all from the computer where WikiExplorator is installed on we can do an XML dump of the database and load this using `Wiki.open_XML` (see `Mediawiki::Wiki.open_XML` for details).[39] Besides reading from a file instead of connecting to the database `Wiki.open_XML` is very similar to `Wiki.open`, all the parameters also apply here.

WikiExplorator tries to be very tolerant regarding missing fields in the database/XML file. As long as the XML file has the right structure and the most important fields are present everything will work fine. See `mediawiki/db_libxml.rb` in the manual for a description of the file format.

So if you are working with a different wiki engine and can provide an XML output following this structure[40] you will be able to analyze it using WikiExplorator. We even tested this with data from other domains not connected to wikis at all. As long as there is something you can call a page, something you can call an user and something you can call a revision which connects pages to users and has a creation time chances are good that it will work.

---

[36] `http://tamsys.sourceforge.net`

[37] I know I should provide a way to give these tables as text files.

[38] ask your local SQL guru how to do this.

[39] this requires `libxml` to be installed which may not be available on Windows.

[40] or create a database

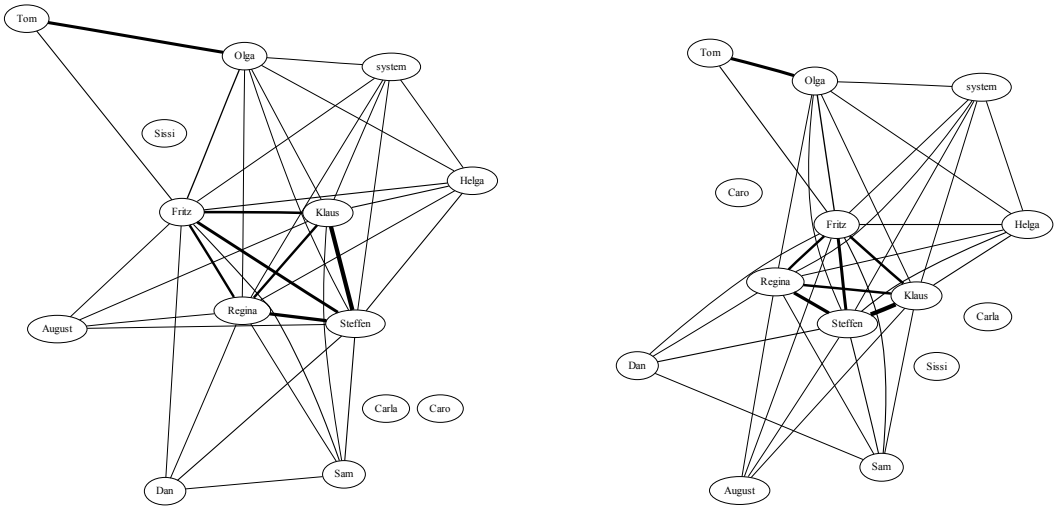


Figure 23: coauthorgraphs without and with edge weights used for layout

8 More Graphs

Until now we only used two simple graphs, the coauthorgraph and the pagegraph. In this section we will have a deeper look on different graphs.

8.1 Edge Weights

As we could see in figures 14 and 15 graphs are by default created with weighted edges. We can use these edge weights in different ways. Some of the graphviz layout engines can use weight information to compute node positions:

```
irb(main):034:0> cgraph = wiki.coauthorgraph; nil
=> nil
irb(main):117:0> cgraph.to_graphviz('gv_ca-noweights.pdf', 'neato', :pdf,
                    'overlap=false', 'splines=true') { |weight|
                                        ["penwidth=#{weight**0.5}", "len=4"] }
=> #<IO:0xf655be7c>
irb(main):118:0> cgraph.to_graphviz('gv_ca-weights.pdf', 'neato', :pdf,
                    'overlap=false', 'splines=true') { |weight|
                         ["penwidth=#{weight**0.5}", "len=#{4/weight**0.25}"] }
=> #<IO:0xf654cb20>
```
54

The `neato` layout engine takes "`len`" as an edge attribute and tries to lay out the nodes in a way that all edges are close to these lengths. We used this here to drag intense coauthors close together, as you can see in figure 23 (right). Additionally we exposed the edge weights by changing the pen width. We could have accomplished a similar effect by changing the edge color from grey to black using the weight.

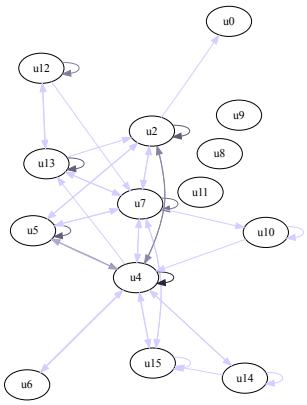


Figure 24: directresponse

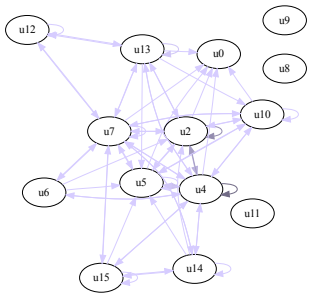


Figure 25: groupresponse

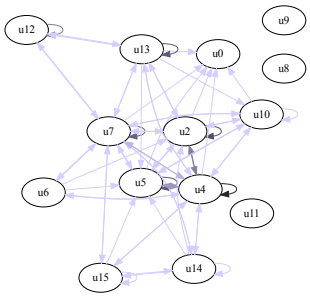


Figure 26: interlocking

8.2 Graph Generation Methods

You may have noticed by reading `Mediawiki::Wiki.coauthorgraph` in the method documentation that the edge weights can be computed in two different ways. By default coauthorship edges are weighted by the number of pages two users share, but you may prefer the Newman measure (see [New01a, New01b]).

The copagesgraph connects users by the pages they edit but does not use the linear structure of the page history for more detailed measures. You may interpret the edit history of a page as a kind of dialogue between users: each edit is an answer to the edit before. This is done by the `Mediawiki::Wiki.directresponsegraph`. It gives a rather sparse graph with directed edges, as two users are only connected if they have successive edits on a page. Alternatively we can use `Mediawiki::Wiki.groupresponsegraph`. Here an edit is considered as answer to all precedent edits, which gives a directed graph similar to the coauthorgraph.

The direct response graph is too picky and the group response graph is too coarse and both do not reflect the page history structure very well. So we developed the interlocking response graph which lays kind of in between (see [SB09, BS07] for a detailed discussion).

```
irb(main):241:0> wiki.directresponsegraph { |n| n.uid }.to_graphviz('gv_drg.pdf',
                                 'neato', 'pdf', 'overlap=false', 'splines=true',
                                 'edge [len=2]') { |w|
                                                 ["color=\"0.7 0.2 #{4/w**0.5}\""] }
=> #<IO:0xf65fcc3c>
irb(main):242:0> wiki.groupresponsegraph { |n| n.uid }.to_graphviz('gv_grg.pdf',
                                 'neato', 'pdf', 'overlap=false', 'splines=true',
                                 'edge [len=2]') { |w|
                                                 ["color=\"0.7 0.2 #{4/w**0.5}\""] }
=> #<IO:0xf655f52c>
irb(main):245:0> wiki.interlockingresponsegraph { |n| n.uid }.to_graphviz(
                 'gv_ilrg.pdf', 'neato', 'pdf', 'overlap=false', 'splines=true',
                                 'edge [len=2]') { |w|
                                                 ["color=\"0.7 0.2 #{4/w**0.5}\""] }
=> #<IO:0xf64e0074>
```

55

Group response and interlocking have the same links but different link weights.[41]

All three methods take various parameters to finetune how exactly the link weights are computed (see the given citations for a discussion of these). interlocking can also be used to create animations using `Mediawiki::Wiki.timedinterlockingresponsegraph`. We will discuss this in section 8.5.

The group response graph and the interlocking graph visually do not differ much from the coauthorship graph. Their strength lies in the statistical measures as the following example shows:

```
irb(main):256:0> wiki.interlockingresponsegraph(:counts => :max).
                                    pp_degrees(:counts => true, :sortby => 1)
Node                          :  deg  out   in
Steffen                       :  145   71   74
Regina                        :   84   40   44
Fritz                         :   73   38   35
Klaus                         :   66   33   33
Olga                          :   48   25   23
Tom                           :   39   20   19
Sam                           :   22   13    9
Dan                           :   17    9    8
Helga                         :   16    8    8
...
=> nil
irb(main):257:0> wiki.interlockingresponsegraph(:counts => :page).
                                    pp_degrees(:counts => true, :sortby => 1)
Node                          :  deg  out   in
Steffen                       :  241  123  118
Klaus                         :  156   74   82
Fritz                         :  111   56   55
Regina                        :  100   49   51
Olga                          :   71   41   30
Tom                           :   54   24   30
Helga                         :   15    6    9
Sam                           :   14    9    5
Dan                           :   10    6    4
...
=> nil
```

56

While Regina has a higher collaborative depth Klaus wins for collaborative breadth (ok, Steffen outperforms all others).

The reverse concept to coauthorship is the copages graph. While the former connects authors by pages the latter connects pages by authors, i. e., two pages are linked if they have at least one common author.

We now have two page graphs: the hyperlink pagegraph showing which pages are explicitly linked by the users and the copages graph connecting pages which have at

[41] You may have to adapt color computation for your wiki. To get best results we could have asked the graph for the maximum link weight.

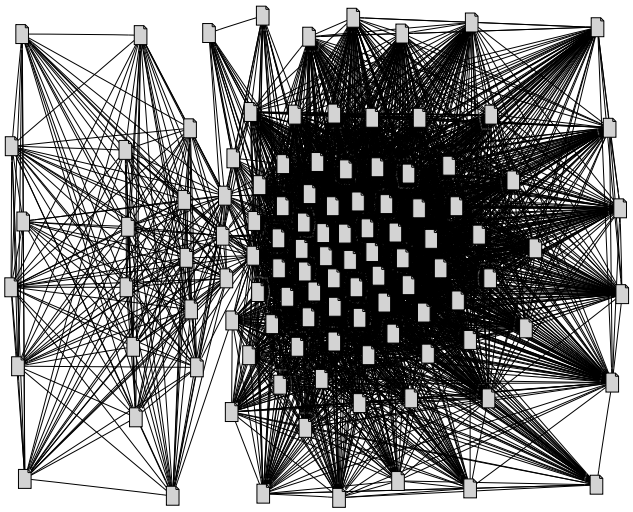


Figure 27: copages graph

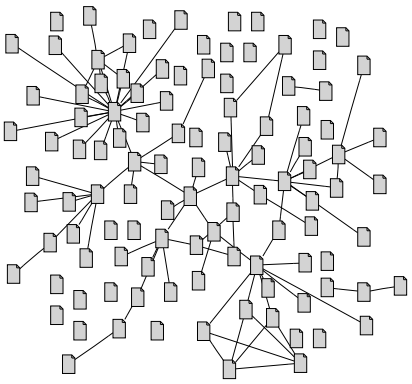


Figure 28: links−copages

least one common author:

```
irb(main):064:0> wiki.copagesgraph {''}.to_graphviz('copages.pdf', :neato, :pdf,
                                   'outputorder=edgesfirst', 'overlap=false',
                                   'node [shape=note, style=filled, width=.34]')
=> #<IO:0xf6701ad8>
```

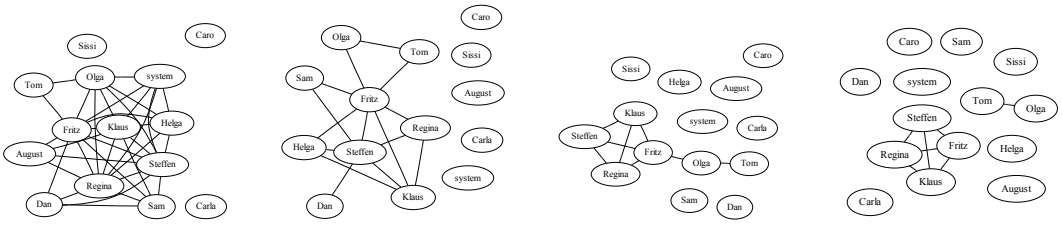
57

We see that while most pages are well connected there is kind of a cluster to the left with only little connection to the rest. We could add node labels or create an SVG with tooltips to see which pages do the connection. Let's go one step further:

```
irb(main):111:0> pg = wiki.pagegraph; nil
=> nil
irb(main):112:0> copg = wiki.copagesgraph; nil
=> nil
irb(main):113:0> pg_copg = DotGraph.new(pg.nodes) {''}; nil
=> nil
irb(main):114:0> pg.links.each { |a,l| pg_copg.link(*a) unless copg.links[a] };
                                                                          nil
=> nil
irb(main):115:0> pg_copg.to_graphviz('pg-copg.pdf', :neato, :pdf,
                                   'outputorder=edgesfirst', 'overlap=false',
                                   'node [shape=note, style=filled, width=.34]')
=> #<IO:0xf66f2754>
```

58

Here we created a *new* graph with pages as nodes and set an edge between two nodes if they are connected in the page graph and not connected in the copages graph (we kind of substract the copages from the page graph). This gives us the pages which are explicitly linked in the wiki but are edited by different groups of users.



full $w > 1$ $w > 2$ $w > 3$

Figure 29: edge weight filtered coauthor graph

8.3 Edge Weight Filters

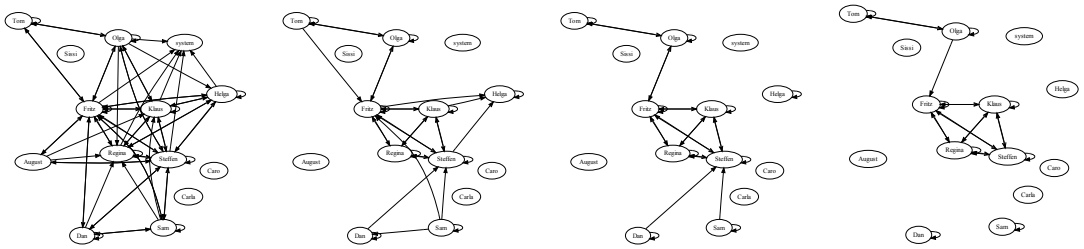
We learned before how to use edge weights for graph layout. One way to visualize connection strength is edge filtering: we remove all edges below a certain weight and show the remaining graph (figure 29):

```
irb(main):264:0> cgraph = wiki.coauthorgraph; nil
irb(main):265:0> params = ['neato', :pdf, 'overlap=false', 'splines=true']
irb(main):266:0> cgraph.to_graphviz('gv_ca0.pdf', *params)
=> #<IO:0xf6518528>
irb(main):267:0> cgraph.remove_links(1.1); nil
=> nil
irb(main):268:0> cgraph.to_graphviz('gv_ca1.pdf', *params)
=> #<IO:0xf650ddf8>
irb(main):269:0> cgraph.remove_links(2.1); nil
=> nil
irb(main):270:0> cgraph.to_graphviz('gv_ca2.pdf', *params)
=> #<IO:0xf6503678>
irb(main):271:0> cgraph.remove_links(3.1); nil
=> nil
irb(main):272:0> cgraph.to_graphviz('gv_ca3.pdf', *params)
=> #<IO:0xf64f9808>
```
59

You get a clear picture about who is working together, and you cannot only print these graphs, you can also apply network measures on them etc. (did you like my nice trick with the additional `params` variable?)

The only drawback may be that the nodes change their position from one step to the other, as the layout manager renders the network with less edges different. So what I would like to do is to render the graph once (with all edges), fix the node positions and remove some edges and render it again. And we can do this by computing node positions once and using them for all four graphs as the following example will show, now featuring the interlocking response graph:[42]

[42] please note that `render_graphviz` may trigger a graphviz bug if you decided to use `dotgraph-gv` and your graphviz version is older than 2009-8-7.



full $w > 1$ $w > 2$ $w > 3$

Figure 30: edge weight filtered interlocking graph with fixed node positions

```
irb(main):014:0> ilgraph = wiki.interlockingresponsegraph; nil
=> nil
irb(main):015:0> gparams = ['overlap=scale', 'splines=true', 'edge [len=3]']
=> ["overlap=scale", "splines=true", "edge [len=3]"]
irb(main):016:0> nodepos = ilgraph.render_graphviz(:neato, *gparams)
=> {"u12"=>"30,482", "u5"=>"242,196", "u13"=>"245,445", "u6"=>"61,178", ...}
irb(main):017:0> ilgraph.nodeblock { |n| ["label=\"#{n.name}\"",
                                          "pos=\"#{nodepos[ilgraph.nid(n)]}\"",
                                          'pin'] }
=> #<Proc:0xf6699474@(irb):17>
irb(main):018:0> ilgraph.to_graphviz('gv_il0.pdf', :nop, 'pdf', *gparams)
=> #<IO:0xf6694398>
irb(main):021:0> ilgraph.remove_links(1.1); nil
=> nil
irb(main):022:0> ilgraph.to_graphviz('gv_il1.pdf', :nop, 'pdf', *gparams)
=> #<IO:0xf6662de8>
irb(main):023:0> ilgraph.remove_links(2.1); nil
=> nil
irb(main):024:0> ilgraph.to_graphviz(DPATH+'gv_il2.pdf', :nop, 'pdf', *gparams)
=> #<IO:0xf6656840>
irb(main):025:0> ilgraph.remove_links(3.1); nil
=> nil
irb(main):026:0> ilgraph.to_graphviz(DPATH+'gv_il3.pdf', :nop, 'pdf', *gparams)
=> #<IO:0xf664a52c>
```
60

`render_graphviz` returns a hash of node positions, and all we have to do now is to tell the `nodeblock` to add these to the corresponding node. And now we use `nop` as layout engine, i. e. a special version of `neato` which respects how to deal with precomputed node positions[43] to print the graph with more and more edges removed. We could even `remove_lonely_nodes` on the graph and the remaining ones would keep their positions.

[43] to be more precisely `nop` takes node positions to be given in points and not in inches as graphviz by default assumes inches as input unit and points ($= \frac{1}{72}$ inch) as output unit, weird enough, and it requires every node to have an initial position set.

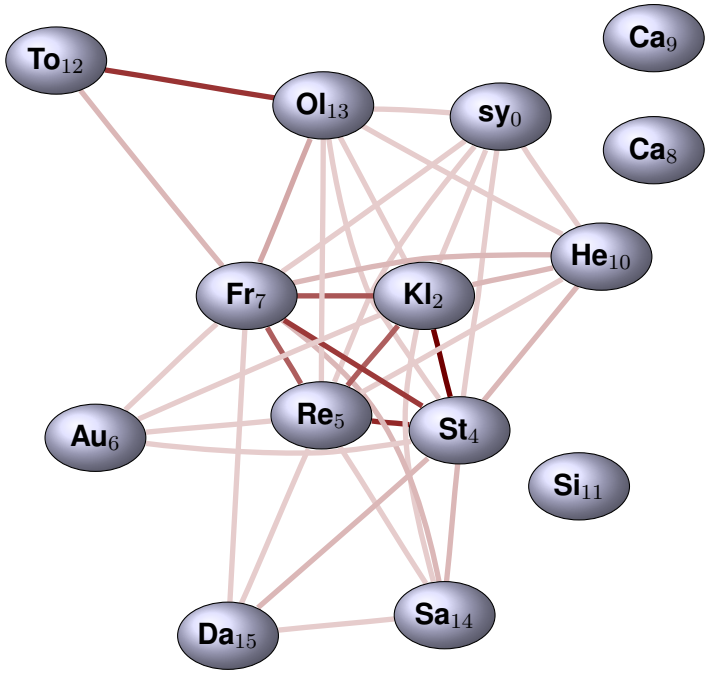


Figure 31: Graphviz, LaTeX and Tikz working together

8.4 LaTeX and Tikz

Graphviz output is really nice. And there is even a way to improve the quality of the rendering if you use LaTeX (and Tikz) to produce your high quality documents (well, I use it for nearly every document, including this tutorial), and if you have `dot2tex`[44] installed. If you want to stay away from the world of good and fast typography and have no connection to LaTeX you may simply skip this section.

`Dotgraph.to_texfile` uses `graphviz` and `dot2tex` to create a LaTeX-file (figure 31):

```
irb(main):206:0> cgraph.nodeblock { |u|
                                    "\\\\textbf{#{u.name[0..1]}}$_{#{u.uid}}$" }
=> #<Proc:0xf66f7100@(irb):206>
irb(main):207:0> cgraph.to_texfile('gv_ca.tex',:alg => :neato, :fmt => 'tikz',
                     :attrs => ['overlap=false', 'splines=true',
                                'node [fixedsize, style="ball color=blue!15"]',
                                'edge [len=2]']) { |w|
                   ["style=\"color=black!50!red!#{w**0.5*20}, line width=2\""] }
=> #<File:~/mwparser/gv_ca.tex (closed)>
```

61

Using tikz has two main advantages: node and edge labels may contain arbitrary LaTeX-code (used here for bold fonts and subscripted uids) and all graphical elements can use the full power of Tikz for rendering[45] (see our shiny nodes).

[44] http://www.fauskes.net/code/dot2tex/

[45] have a look in the `dot2tex` (http://www.fauskes.net/code/dot2tex/gallery/) and the Tikz/PGF (http://www.texample.net/tikz/examples/) galleries for impressing examples.

8.5 Graph Animations

I promised you in section 8.2 to tell you more about how to create nice animantions. The method `Mediawiki::Wiki.timedinterlockingresponsegraph` gives an interlocking response graph of our wiki where each response is denoted by an explicit link attributed by the time stamp of the response. Dotgraph is able to create SoNIA[46] animation source files if the graph contains edges attributed with timestamps. So all we have to do is:

```
irb(main):063:0> wiki.timedinterlockingresponsegraph.to_sonfile('il.son')
=> #<File:~/mwparser/il.son (closed)>
```
62

`to_sonfile` takes various parameters which allow to finetune node and edge appearance in the animation. The defaults should give you a good starting point.

Now we can start SoNIA, load `il.son` and create nice animations like the ones you can see at `http://www.kinf.wiai.uni-bamberg.de/mwstat/Inhalt.html`. How to use SoNIA is beyond this tutorial, sorry.

Creating son-files certainly works on any graph if you provide timed links.

9 Writing Scripts

Up to now we used WikiExplorator interactively with `irb`. Ruby is a full-featured programming language, so we may want to write some ruby programs for our work. So I use my favourite editor to open `myscript.rb`:

```
#!/usr/bin/ruby -w
require 'mediawiki/full'  # load all parts of the WikiExplorator library
# now we open our wiki:
wiki = Mediawiki::Wiki.open("wiodb", "wiowiki.kinf.wiai.uni-bamberg.de",
                            "wiouser", IO.getpw,
                            :language => 'de', :name => 'WiO')
wiki.pp_global_userstats  # and some output. More should follow
```
63

The first line tells unixoide platforms that this is a ruby script. Then we require the `mediawiki/full` library and now we can open our wiki. I did not save the password in the script but ask the user to type it in using `IO.getpw`. And finally I print ouzt some statistics.

Alternatively we could have used the startup file we created before:

```
#!/usr/bin/ruby -w
require 'wio'                              # load the startup file
wiki = Mediawiki.mywiki(IO.getpw)          # open the wiki
wiki.pp_global_userstats                   # some output
puts "My personnel wiki statistic tool!" # Hurray!
```
64

So, go and learn some Ruby, write nice methods for sophisticated statistics and visualization and send them to me to be integrated in the codebase.

[46] `www.stanford.edu/group/sonia/`

10 Customized Reports

Besides interactive usage and specially written programs WikiExplorator is also able to do nice wiki reports as PDF or HTML:[47]

```
irb(main):117:0> Mediawiki::Report.new(wiki, :pdf, :template => 'web',
                                       :outputdir => 'mw-report').generate
Reading './mediawiki/reports/web.de.tex' failed.
Trying './mediawiki/reports/web.tex'.
Found some NaN's! Removed!
This is pdfTeXk, Version 3.141592-1.40.3 (Web2C 7.5.6)
 %&-line parsing enabled.
entering extended mode
=> "~/mwparser/mw-report/default.pdf"
```
65

This generates a report with template "`web`" in the output directory "`mw-report`". As you can see in the next line WikiExplorator first tries to find a report file with the right language ("`de`" for our example wiki) and falls back to the default file. And finally the path of the report is returned.

We may want to create an own report customized to our needs. Let's create our own HTML report by creating a new file "`mediawiki/reports/myreport.html`" (figure 32). It is an ordinary HTML file with ruby commands embedded (this is similar to PHP). Embedded commands are enclosed in `<%...%>` or `<%=...%>`. The latter form includes the return value of the command in the resulting HTML file.

As you can see we use `@wiki` to access our wiki. The "`@`" indicates ruby instance variables and within the report file WikiExplorator provides the wiki to us in the instance variable `@wiki`.

So what we are doing here is to create a second filter ("`full`") including all namespaces, set some local variables and write some text with embedded commands giving simple descriptive statistics about the wiki. Then we create some nice graphic and point to it using the "`img`" tag.

Try it:

```
irb(main):126:0> Mediawiki::Report.new(wiki, :html, :template => 'myreport',
                                       :outputdir => '/tmp/mw-report').generate
Reading './mediawiki/reports/myreport.de.html' failed.
Trying './mediawiki/reports/myreport.html'.
=> "~/mwparser/mw-report/default.html"
```
67

As you can see first the language specific file was tried and as it was missing WikiExplorator fell back to the default.

Now point your webbrowser to "`~/mwparser/mw-report/default.html`" and see the result (figure 33).

Creating LaTeX/PDF templates works the same way. Use the files in "`mediawiki/reports/`" as starting point for own reports, but save them under new names (otherwise you would

[47] The same command line is also used in our startup file.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>My report: <%= @wiki.to_s %></title>
  </head>
  <% full = @wiki.filter.clone; full.include_all_namespaces %>
  <% wiki.filter.namespace=0 %>
  <% users = @wiki.users %>
  <% pages = @wiki.pages %>
  <% revisions = @wiki.revisions %>
  <body>
    <h1>My report: <%= @wiki.name %></h1>
    <p>
      We have <%=@wiki.pages(full).length%> pages with
      <%=@wiki.revisions(full).length%> revisions from
      <%=users.length%> users (<%=pages.length%> pages with
      <%=revisions.length%> revisions in Namespace 0).
    </p>
    <% up = users.collect { |u| u.pages.length }
       up.gp_plot_lorenz(:title => "Lorenz Curve", :xlabel => "authors",
                         :ylabel => "pages", :png=>'cdfap.png',
                         :size => '480,480')
    %>
    <p>
      <img alt="lorenz curve - pages vs. authors" src="cdfap.png" />
    </p>
  </body>
</html>
```

66

Figure 32: mediawiki/reports/myreport.html

My report: WiO

We have 271 pages with 1362 revisions from 14 users (109 pages with 1125 revisions in Namespace 0).

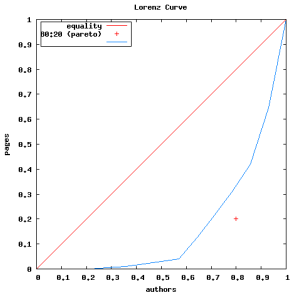


Figure 33: output for myreport.html

loose then on update).

And do not forget to send them to me for inclusion in the WikiExplorator package, I am interested in new interesting reports, in translations of reports to other languages etc., in all kinds of improvements.

11 Acknowledgements

Wiki Explorator[48] was developed[49] by

Dr. Klaus Stein
Laboratory for Semantic Information Technology
Chair of Computing in the Cultural Sciences
Otto-Friedrich Univerity Bamberg

and is available as open source.

References

[BS07] Steffen Blaschke and Klaus Stein. Methods and measures for the analysis of corporate wikis. In *Proceedings of the 59th Annual Conference of the International Communication Association (ICA)*, 2007.

[New01a] Mark E. J. Newman. Scientific collaboration networks. i. network construction and fundamental results. *Physical Review E*, 64(1):016131, Jun 2001.

[New01b] Mark E. J. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical Review E*, 64(1):016132, Jun 2001.

[SB09] Klaus Stein and Steffen Blaschke. Collaborative intensity in social networks. In *Proceedings of the International Conference on Advances in Social Network Analysis and Mining*, 2009.

[48] `http://wiki-explorator.rubyforge.org`
[49] in the WiO project supported by a grant from the Volkswagen Stiftung.