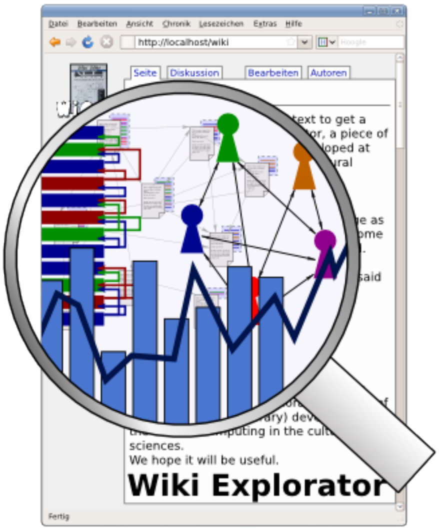
WikiExplorator Beginners Tutorial

Klaus Stein

September 1, 2009


Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://localhost/wiki
Seite
Diskussion
Bearbeiten
Autoren
sciences.
We hope it will be useful.
Wiki Explorator
Fertig

Contents

1 Introduction

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[1] and experiment with different parameter sets.

2 Installation

I unpacked the archive to the directory `mwparser` and installed Ruby[2], 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

Using the class and method documentation[1] 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

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

[2] 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.

to `wiodb`, give the database host, the database user and the name and language of the wiki[3]:

```
  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`[4].

So, let's see if everything works:[5]

```
~/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.

[3] if our tables have some prefix "`xxx`", we would add `:prefix => 'xxx'` within the braces.

[4] 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).

[5] 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:[6]

```
~/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.[7]

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

[7] 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 a block as a parameter. A block is a piece of code given in braces[8] 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):

[8] 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.

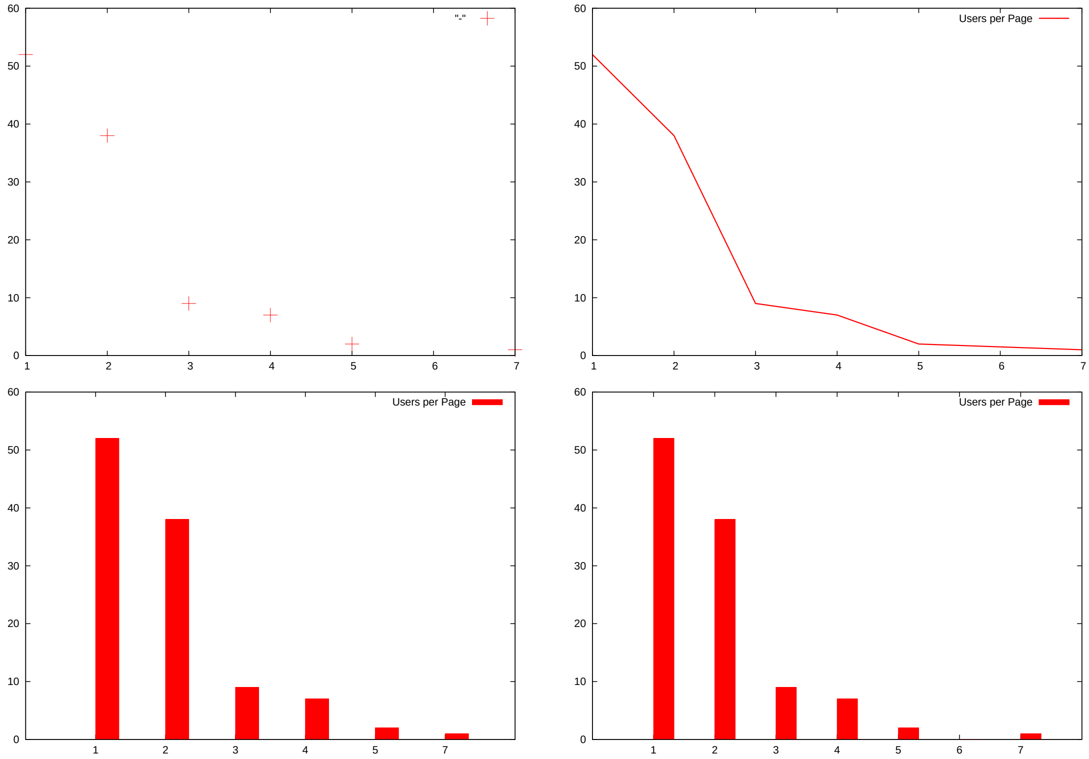


Figure 1: Using Gnuplot for data presentation

4 Gnuplot

Let's use gnuplot for a graphical representation of this histogram:[9]

```
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):

[9] using pu as defined in session 15

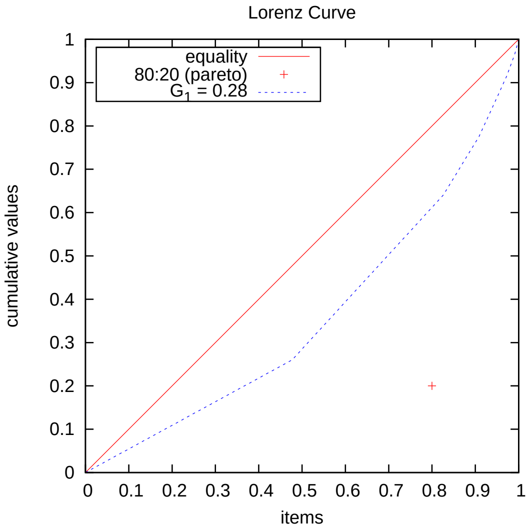


Figure 2: Lorenz curve for users per page

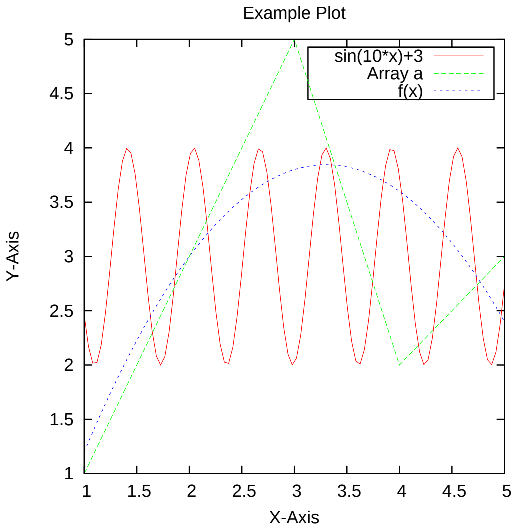


Figure 3: Extended gnuplot example

```
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[10], 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 $\sin(10x) + 3$:

[10] `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[11]:

```
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

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

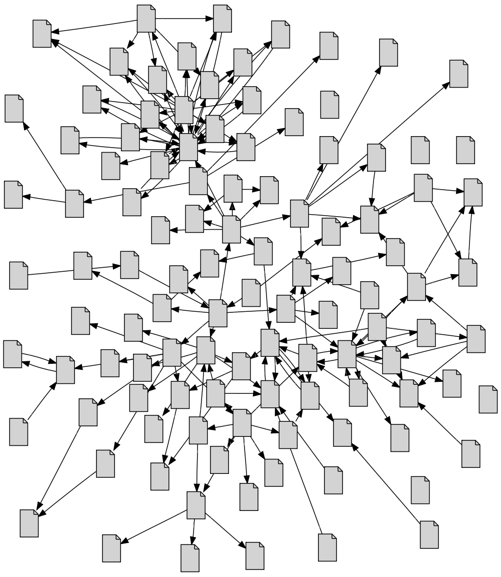


Figure 4: hyperlink network

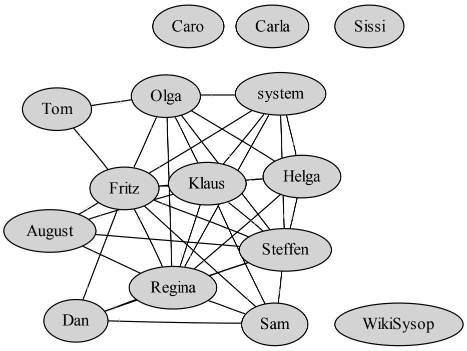


Figure 5: coauthorship network

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.[12]

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 ...>
```

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):

[12] 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[13]
=> 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.

[13] 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 section ??):

```
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.

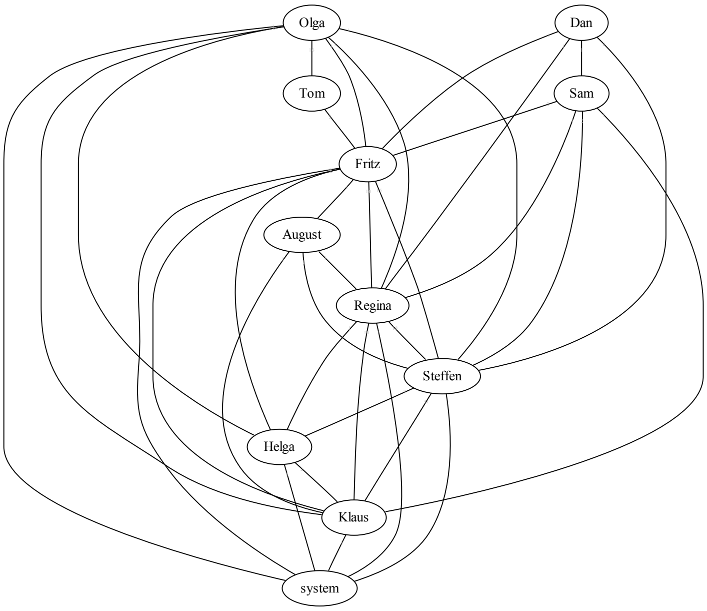


Figure 6: `dot` layout

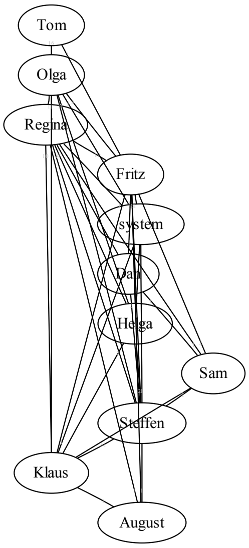


Figure 7: `fdp` layout

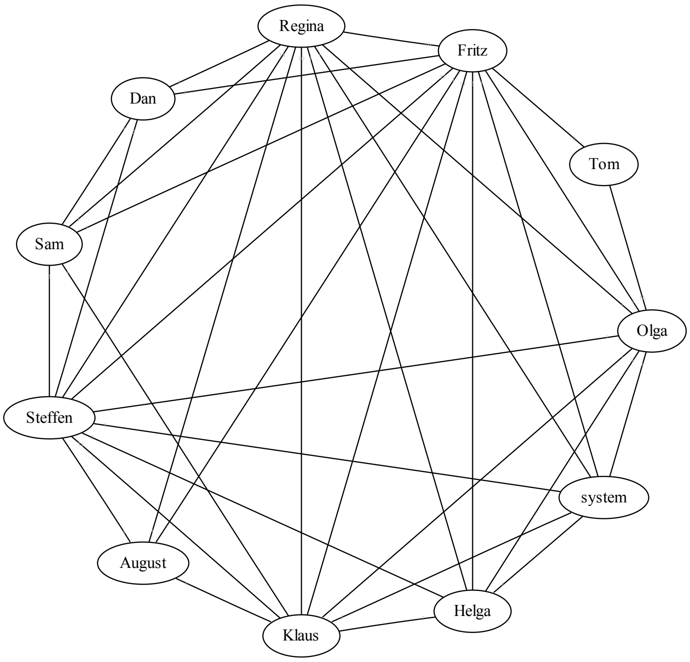


Figure 8: `circo` layout

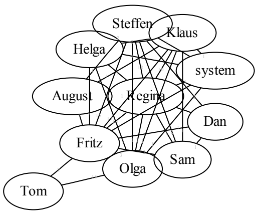


Figure 9: `twopi` layout

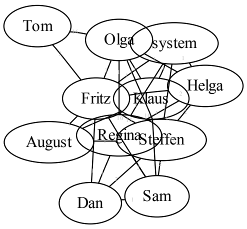


Figure 10: `neato` layout

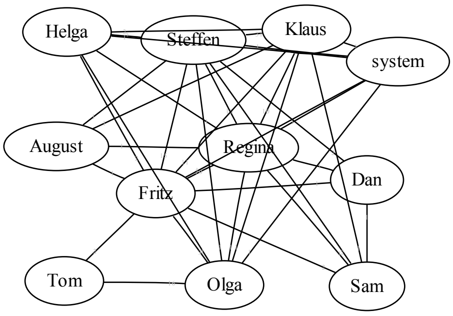


Figure 11: `twopi` without overlap

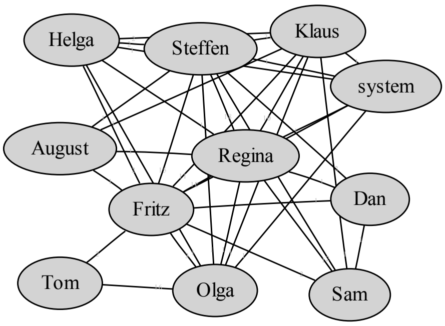


Figure 12: `twopi`: nodes in front of edges

5.3.1 Graphviz

`graphviz` can be used in two ways: if `ruby-graphviz` (module `gv`) is present it is used, otherwise it is silently substituted by calls to the `graphviz` executables.[14] We start by using the different graphviz layout engines[15] and create some PDFs:[16]

```
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.[17] 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
```

[14] the main drawback here is that it is slow.

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

[16] (you may need to use `:pspdf` instead, see footnote 11, Sec. 4), others (SVG, PNG, . . .) are available.

[17] 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.

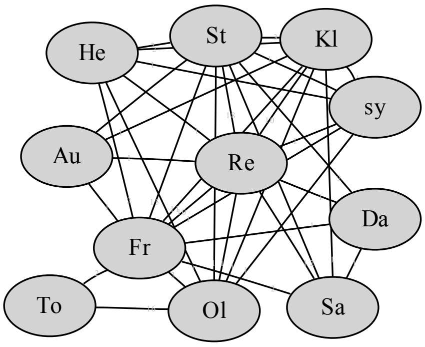


Figure 13: `twopi`: short labels

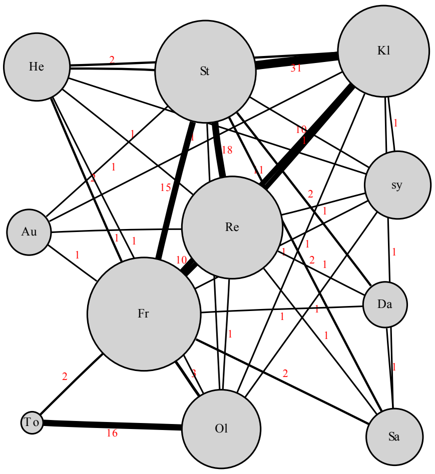


Figure 14: `twopi`: weighted sizes

The first call[18] gives figure 11, the second[19] 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):

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

[19] `'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[20]:

```
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.[21] 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.[22] 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,[23], 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.

[20] coauthorgraph weights edges between authors with the number of pages they are coauthors on.

[21] the divisor has to be a Float, otherwise ruby would use Integer division

[22] 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.

[23] see the gallery at http://www.graphviz.org/Gallery.php

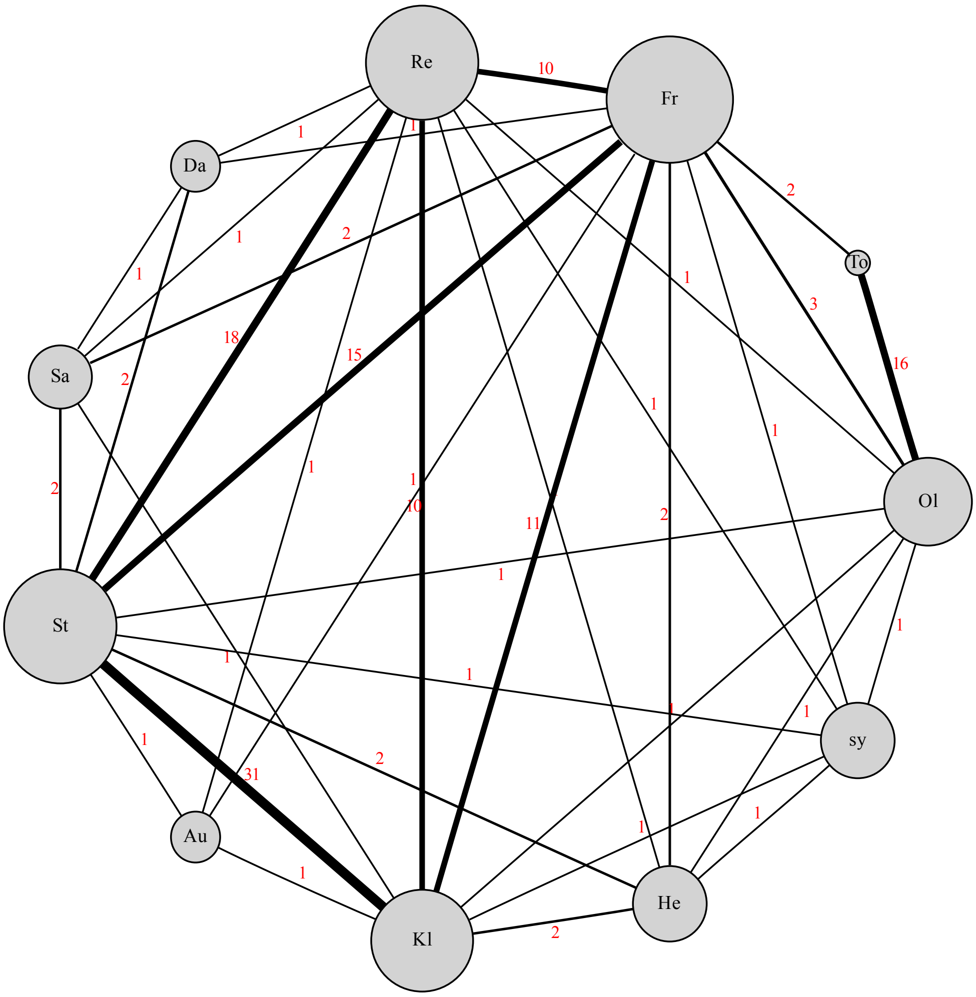


Figure 15: `circo`: weighted sizes

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

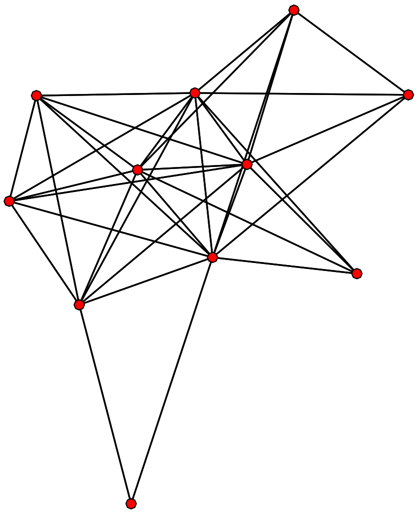


Figure 16: Plotting using R

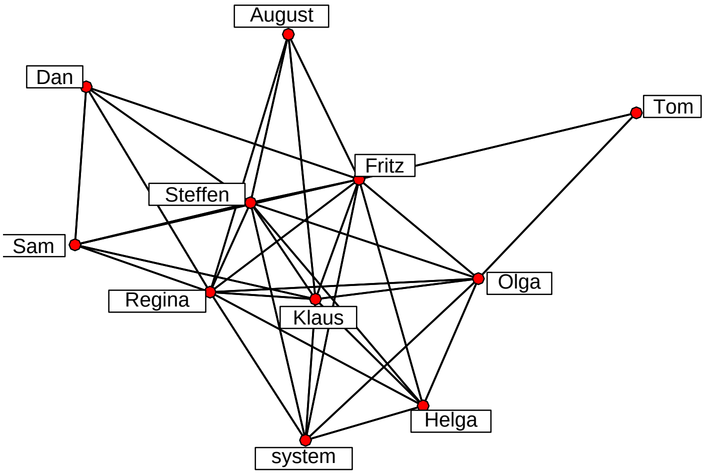


Figure 17: Plotting with labels using R

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):[24]

```
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.

[24] 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”.[25] 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.

[25] 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,[26] after that we restrict the namespace of f2 to 0 and then we use f2 for the revisions view. Any method of Wiki returning a view as well as many of the other methods[27] take a filter as optional parameter.[28]

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, and we do this using filters.

[26] cloning is important, otherwise both varfiables would point to the *same* object!
[27] i. e. graph generation and statistics methods
[28] send a bug report if I missed a method.

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

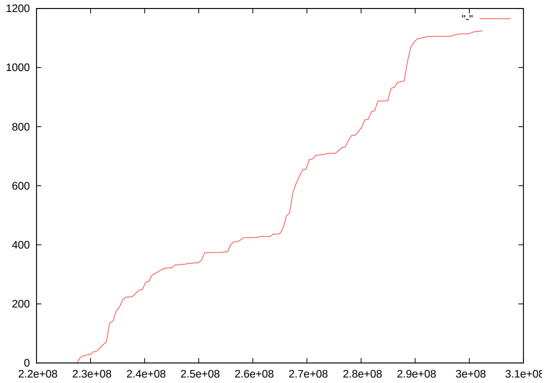


Figure 18: weekly revisions (cumulative)

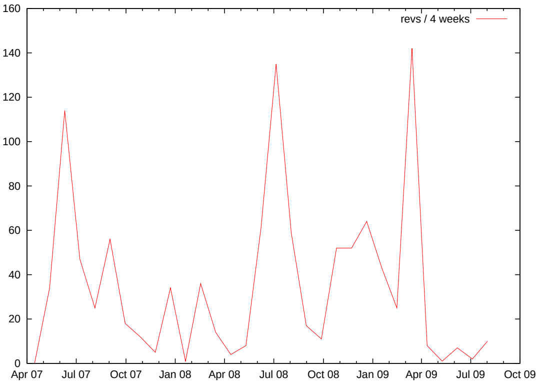


Figure 19: monthly revisions

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 ToDo

- user aliasing
- Full R access (manual, `r.rb`)
- further graph generation methods
- sonia
- loading and dumping (marshal, yaml), obliterate
- more sophisticated examples
- scripting
- customized reports (html, LaTeX)
- conclusion/summary/overview/architecture