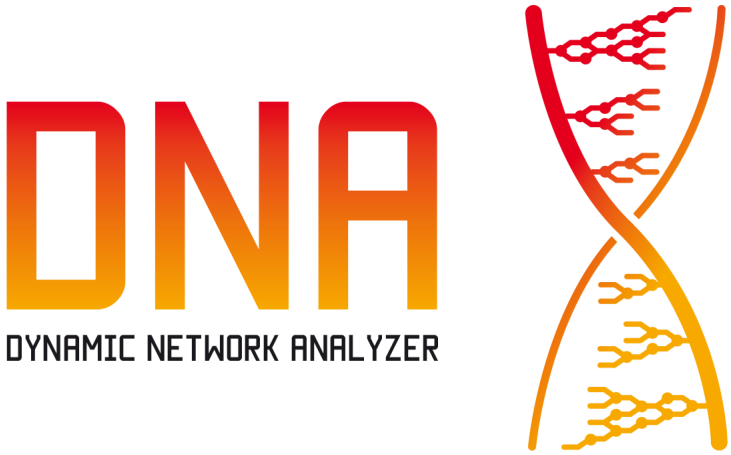


Plotting
Instruction Manual

René Wilmes

25.04.2015

Dresden University of Technology

Contents

Chapter 1

Introduction

The DNA – Dynamic Network Analyzer is a framework for Graph-theoretic Analysis of Dynamic Networks. In order to illustrate and compare generated data, DNA is equipped with numerous features to plot data in a sophisticated way, while still allowing the user to control the plotting process and to customize the plots visual representation aswell as creating his own custom plots.

Throughout this manual the numerous plotting features will be described aswell as their respective use explained. Examples give the user an insight on the impact of certain customizations.

The manual is divided into two parts. The first addresses the plotting process in general, what methods to choose and what parameters to set in order to reach the desired results. The second part demonstrates how to create plots, what possibilities and limits there are and how to handle custom plots in general.

Chapter 2

Plotting in DNA

This chapter will deal with the general plotting mechanisms in DNA and how to use them. All plots are created by using gnuplot.

2.1 What can be plotted?

In general it is possible to plot all values that are being gathered during generation. This covers runtimes and statistics aswell as data contained in metrics, namely values, distributions and nodevaluelists.

2.2 How to plot?

The plotting class dna.plot.Plotting.java holds several public methods, which can be called to start the plotting process. Overloading allows to call these methods either with a single SeriesData object or with an array of these. It is either possible to plot aggregated data of several runs, or the non-aggregated runs themself. Note that the process of comparing and plotting multiple series differs from plotting only a single series. Therefore, if only one series is handed over, the single plotting process will be performed. For more than one series the multiple plotting process will be executed respectively. For more details see section 2.5 Plotting methods.

2.3 PlottingConfig

The PlottingConfig class provides a way for the user to easily configure the plotting process. A PlottingConfig object can be instantiated, configured and then handed over to a plotting method. It holds several methods to customize the plotting, for example set a timestamp window and stepsize for the plots. Furthermore PlotFlag objects can be handed over to partly enable the plotting of choosen data, instead of plotting the whole series. See table 2.1 for more details.

Method	Description
setDistPlotType(DistPlotType d)	Sets if distributions will be plotted normally, as cdf or both.
setNvlOrder(NodeValueListOrder n)	Sets if the nodevaluelist‘s will be ordered, ascending or descending
setNvlOrderBy(NodeValueListOrderBy n)	Sets which value will be used for the ordering, for example ordering by index or by average.
setPlotCustomValues(boolean b)	Enables/disables custom value plots.
setPlotDistributions(boolean b)	Enables/disables distribution plots.
setPlotMetricValues(boolean b)	Enables/disables metric value plots.
setPlotNodeValueLists(boolean b)	Enables/disables nodevaluelist plots.
setPlotRuntimes(boolean b)	Enables/disables runtime plots.
setPlotStatistics(boolean b)	Enables/disables statistic plots.
setPlotInterval(long from, long to, long stepsize)	Sets a plot interval. Will only plot batches whose timestamps are inside the interval.
setPlotIntervalByIndex(int from, int to, int stepsize)	Sets a plot interval. Will only plot batches whose indices are inside the interval.
setPlotStyle(PlotStyle s)	Sets the plot style of all plots. For example: linespoint, dots, points and more.
setPlotType(PlotType t)	Sets what values of the data will be plotted. For example: average, minimum...

Table 2.1 – PlottingConfig API methods.

2.4 PlotFlag

PlotFlag objects are used to define what data will be plotted. They can either be handed over to the plotting methods directly or set in a PlottingConfig object. Note that if no flag is handed over, the program will use a plotAll-flag and therefore plot all types of data. Table 2.2 shows all available flags.

2.5 Plotting methods

The following is a list of supported plotting methods in the current build and a brief explanation.

Simple methods which plot the series to the destination directory.
- plot(SeriesData seriesData, String dstDir)
- plot(SeriesData[] seriesData, String dstDir)

Plot only data enabled by the PlotFlag‘s to the destination directory.
- plot(SeriesData seriesData, String dstDir, PlotFlag... flags)
- plot(SeriesData[] seriesData, String dstDir, PlotFlag... flags)

Plot the series to the destination directory and takes the given PlottingConfig object to configure the plotting process.

Flag	Description
plotCustomValues	Enables custom value plots.
plotDistributions	Enables the plotting of distributions.
plotMetricValues	Enables the plotting of metric values.
plotNodeValueLists	Enables the plotting of nodevaluelists.
plotRuntimes	Enables the plotting of runtimes.
plotStatistics	Enables the plotting of statistics.
plotMetricEntirely	Plots all metrics entirely. Has the same effect as the combination of plotDistribution, plotMetricValues and plotNodeValueLists.
plotSingleScalarValues	Plots all single scalar values. Has the same effect as the combination of plotCustomValues, plotMetricValues, plotRuntimes and plotStatistics.
plotMultiScalarValues	Plot all multi scalar values. Has the same effect as the combination of plotDistributions and plotNodeValueLists.
plotAll	Enables all flags.

Table 2.2 – List of PlotFlag's

- plot(SeriesData seriesData, String dstDir, PlottingConfig config)
- plot(SeriesData[] seriesData, String dstDir, PlottingConfig config)

Plot data in the given timestamp window to the destination directory. (Note: If no PlotFlag's are handed over, all data will be plotted.)
- plotFromTo(SeriesData seriesData, String dstDir, long timestampFrom, long timestampTo, long stepsize, PlotFlag... flags)
- plotFromTo(SeriesData[] seriesData, String dstDir, long timestampFrom, long timestampTo, long stepsize, PlotFlag... flags)

Plot data in the given index window to the destination directory. (Note: If no PlotFlag's are handed over, all data will be plotted.)
- plotFromToByIndex(SeriesData seriesData, String dstDir, int indexFrom, int indexTo, int stepsize, PlotFlag... flags)
- plotFromToByIndex(SeriesData[] seriesData, String dstDir, int indexFrom, int indexTo, int stepsize, PlotFlag... flags)

Plot the (non-aggregated) run with index i of the series to the destination directory.
- plotRun(SeriesData seriesData, int i, String dstDir, PlotFlag... flags)
- plotRun(SeriesData seriesData, int i, String dstDir, PlottingConfig config)

Plot all (non-aggregated) runs of the series separately to the destination directory.
- plotRunsSeparately(SeriesData seriesData, String dstDir, PlotFlag... flags)
- plotRunsSeparately(SeriesData seriesData, String dstDir, PlottingConfig config)

Plot all (non-aggregated) runs of the series in combinated plots to the destination directory.
- plotRunsCombined(SeriesData seriesData, String dstDir, PlotFlag... flags)
- plotRunsCombined(SeriesData seriesData, String dstDir, PlottingConfig config)

Plot the (non-aggregated) run with index i from all series in combinated plots to the destination directory.
- plotRun(SeriesData[] seriesData, int i, String dstDir, PlotFlag... flags)
- plotRun(SeriesData[] seriesData, int i, String dstDir, PlottingConfig config)

Plot all (non-aggregated) runs of the series to the destination directory. The i-th run of each series will be combined.
- plotRunsSeparately(SeriesData[] seriesData, String dstDir, PlotFlag... flags)
- plotRunsSeparately(SeriesData[] seriesData, String dstDir, PlottingConfig config)

2.6 Example

The following code snippet shows an example on how to use a PlottingConfig object in order to configure the plotting process.

```
public static void main(String[] args) throws AggregationException,
IOException, MetricNotApplicableException, InterruptedException {
  // init graph- & batch generators
  GraphGenerator gg1 = new RandomGraph(GDS.directed, 100, 300);
  BatchGenerator bg1 = new RandomBatch(0, 0, 10, 5);

  // define metrics
  Metric[] m1 = new Metric[] { new DegreeDistributionR() };

  // create series and generate it
  Series s1 = new Series(gg1, bg1, m1, "data/test1/", "Test1");
  SeriesData sd1 = s1.generate(1, 100);

  // create PlottingConfig object, only plot runtimes
  PlottingConfig config = new PlottingConfig(PlotFlag.plotRuntimes);

  // set plot interval: will plot every 5th batch in [0:100]
  config.setPlotInterval(0, 100, 5);

  // set to plot the median of all aggregated data with lines
  config.setPlotStyle(PlotStyle.lines);
  config.setPlotTyle(PlotType.median);

  // more configuration possible here

  // start plotting
  Plotting.plot(sd1, "data/test1/plots/", config);
}
```

Figure 2.1 – PlottingConfig example.

Chapter 3

Custom Plots

The custom plots form a powerful mechanism allowing the user to freely create and configure additional plots. The focus of this chapter lies on how to create and customize said custom plots.

3.1 Location

All custom plot code should be written to the respective custom-config files located in the dna/config directory. These files hold all default custom plots aswell as extensive examples.

3.2 Syntax

For each type of custom plot there is a default prefix. See table 3.1. For each prefix, there is a field called which defines the respective plots. For example: defines, that there are two custom runtime plots, called R1 and R2. These identifiers will be used further to specify the plots. See the examples in section 3.2.1 for further details.

Prefix	**Type**
CUSTOM_	General custom plot which can contain any sort of values.
RT_	Runtime plot.
ST_	Statistics plot.
MV_	Metric-Value plot.
MD_	Metric-Distribution plot.
MNVL_	Metric-nodeValueList plot.

Table 3.1 – Prefixes for custom plots.

3.2.1 Example

The code snippet below shows an example on how one could specify a custom plot, which will plot the inDegreeDistribution of a metric called DegreeDistributionR as a cumulative distribution function (cdf).

```
MD_PLOTS = D1
MD_1_FILENAME = distributionPlot1
MD_D1_VALUES = DegreeDistributionR~inDegreeDistribution
MD_D1_TYPE = distANDcdf
MD_D1_KEY = off
```

Figure 3.1 – Example of a custom distribution plot.

Note the syntax used in the definition. The prefix MD_ derives from the prefix for metric distribution plots and is followed by the identifier D1, which itself is always followed by a suffix. The suffix _VALUES is the only mandatory field and defines which values will be included in the plot. A list of all available suffixes can be found in table 3.2. The _KEY field is used to configure the legend in gnuplot. Here it is turned off for a clear result.

3.2.2 Domains

In order to adress a value in a unique way, it is neccessary to know where it is stored. For example two different metrics “metric1“ and “m2“ may calculate the number of edges in a graph and store it as a value called “edges“. For this case the domain got introduced. It can either be the name of a metric or a static string. The following shows how the two different values can be adressed:

metric1~edges
m2~edges

Note the delimiter ~. As seen in the former example 3.2.1, the “inDegreeDistribution “ has been adressed with

DegreeDistributionR~inDegreeDistribution ,

where “DegreeDistributionR“ is the domain. Runtimes and statistics can be adressed with static domains. For a list of all staticly defined domains, check table 3.3. Note that for custom plots with the prefix ST_ and RT_ it is not necessary to set domains, as these are statistics or runtime plots by definition and values from other domains are not allowed.

3.3 Wildcards

The described way to define custom plots is very restricted in regards of dynamic use. In each plot the value‘s have to be explicitly defined. There are several cases where this might not be convenient. For example the user wants a plot, which shows the runtimes of all current metrics for arbitrary series‘. Wildcards * provide a sophisticated way to deal with such cases. The code in figure 3.3 creates a plot with the runtimes of all used metrics, in this case DiceDirectedR-out, DicreDirectedU-out and DegreeDistributionR, and additionally the total runtime. Note that wildcards can not be used to replace domains. Additionally only one wildcard might be used in the same value definition.

Suffix	Type
_FILENAME	Filename of the resulting script and plot.
_VALUES	Defines the values contained in the plot.
_DOMAIN	Defines a domain for the plot. If set all values that are being defined without domain will be treated as if they had this domain.
_TITLE	Title of the plot.
_CDF	If the plot should be plotted as cdf. Possible values: true, false, both.
_DATETIME	If set datetime will be used for timestamps on the x-axis. Uses gnuplot datetime.
_LOGSCALE	Sets if x and/or y axis should have logarithmic scaling. Possible values: x, y, xy
_XLABEL	Sets the label of the x-axis.
_YLABEL	Sets the label of the y-axis.
_XOFFSET	Sets an offset for the x-axis.
_YOFFSET	Sets an offset for the y-axis.
_XRANGE	Sets the range of the x-axis.
_YRANGE	Sets the range of the y-axis.
_XTICS	Sets the tics which will be shown on the x-axis. See gnuplot documentation for futher details.
_YTICS	Sets the tics which will be shown on the y-axis. See gnuplot documentation for futher details.
_TYPE	Only available for distribution plots. Sets how the distribution will be plotted.
_STYLE	Style of the plot. Possible values: lines, dots, points, linespoint, impulses, steps, boxes, candlesticks, yerrorbars, fillsteps, filledcurves
_ORDER	Only available for nodevaluelist plots. Sets how the nodevaluelists should be ordered. Possible values: ascending, descending.
_ORDERBY	Only available for nodevaluelist plots. Sets by which value the nodevaluelists should be ordered. Possible values: index, average, median, minimum, maximum, variance, varianceLow, varianceUp, confidenceLow, confidenceUp.

Table 3.2 – List of all suffixes.

Prefix	Type
statistics	Domain of statistical values.
runtimes	Domain of runtimes. Can be used to adress either. metric runtimes or general runtimes.
metric_runtimes	Domain of metric runtimes.
general_runtimes	Domain of general runtimes.

Table 3.3 – List of static domains.

3.4 Mathematical expressions

The modification of plotted data can often be necessary. For example the runtimes are measured in nanoseconds. Mathematical expressions can be used to transform these into seconds. Example 3.4 shows how one could do that. Note the double point : , which signals the DNA that it is dealing with a mathematical expression. Variables are ¡domain¿ ¡value¿ pairs surrounded by $. Note that it is possible to add multiple variables to one expression. This allows to also combine different values. In addition wildcards can be used in mathematical expressions. For each value the wildcard will be replaced by the respective value and then calculated.

The plot in figure 3.4 achieved exactly what we wanted. The runtimes are being devided by 10*9 and therefore appear as seconds. However, there is no label on the y-axis indicating that we are dealing with seconds. By adding a line defining the _YLABEL, we can set it to “sec“, see figure 3.5.

Now the y-axis is labeled and the plot got more reasonable. But still, the lines hold the entire mathematical equations in their labels. In order to clean up the plot we can give the mathematical expression a name by adding a string before the double point. The example code in figure 3.6 assigns “ab c“ as the expressions name.

As we can see, all lines created by the expression have been named “ab c“. However, especially when dealing with wildcards, this behaviour might not be desirable, because for multiple values it is not clear which line represents which value. More sophisticated naming schemes can be created by inserting a wildcard surrounded by $ into the name string. The wildcard will be replaced by the name of the respective value. See figure 3.7 on the next page for an example.

3.5 Functions

The above described mathematical expressions only allow to transform given values. In order to investigate scaling or complexity problems it can be useful to add chosen functions to the plot. This is possible with straight-forward function definitions in the values field. The example in figure 3.8 extends the in the last section introduced mathematical expression example by adding a function. A more sophisticated example of function definitions can be found on the following page.

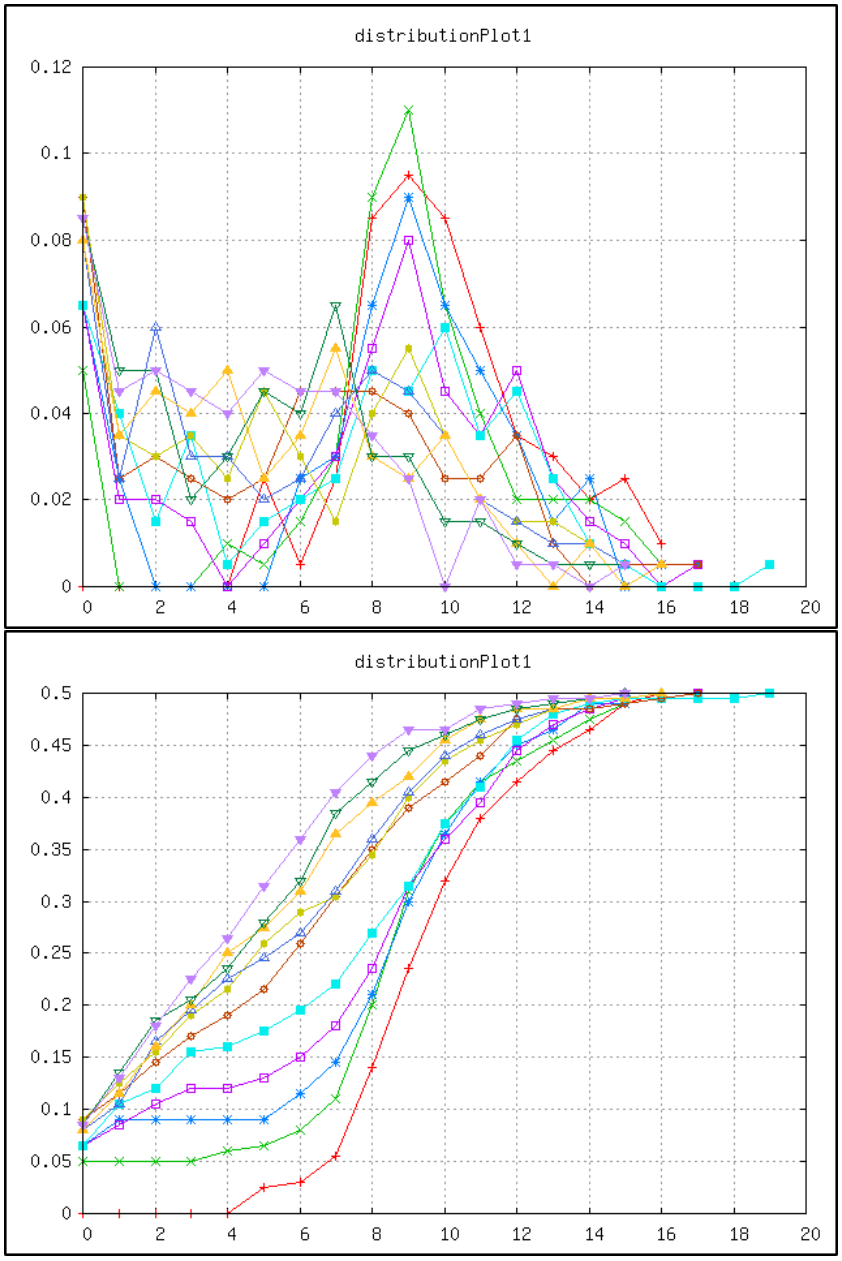
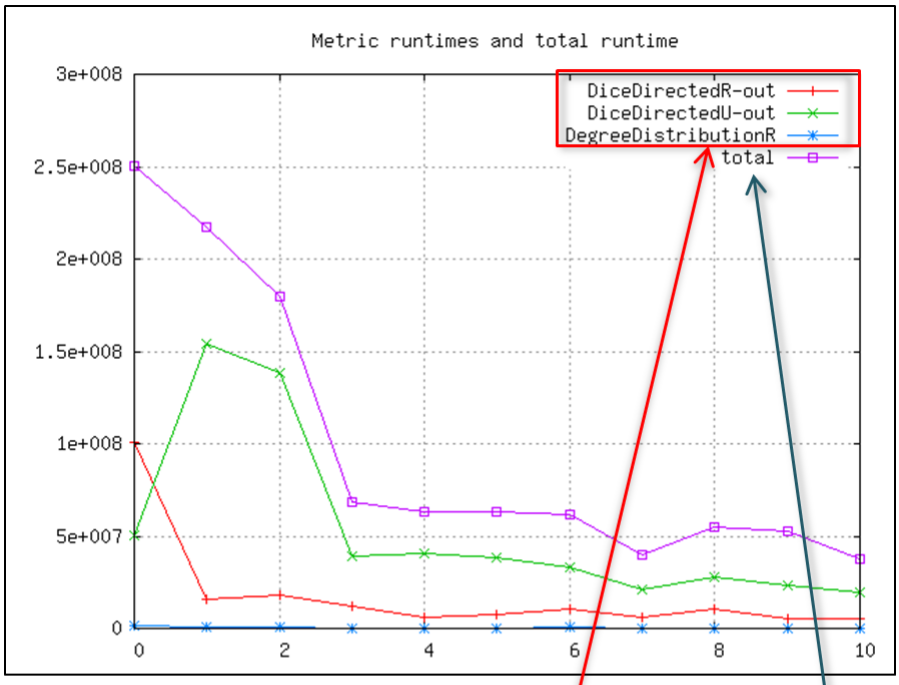
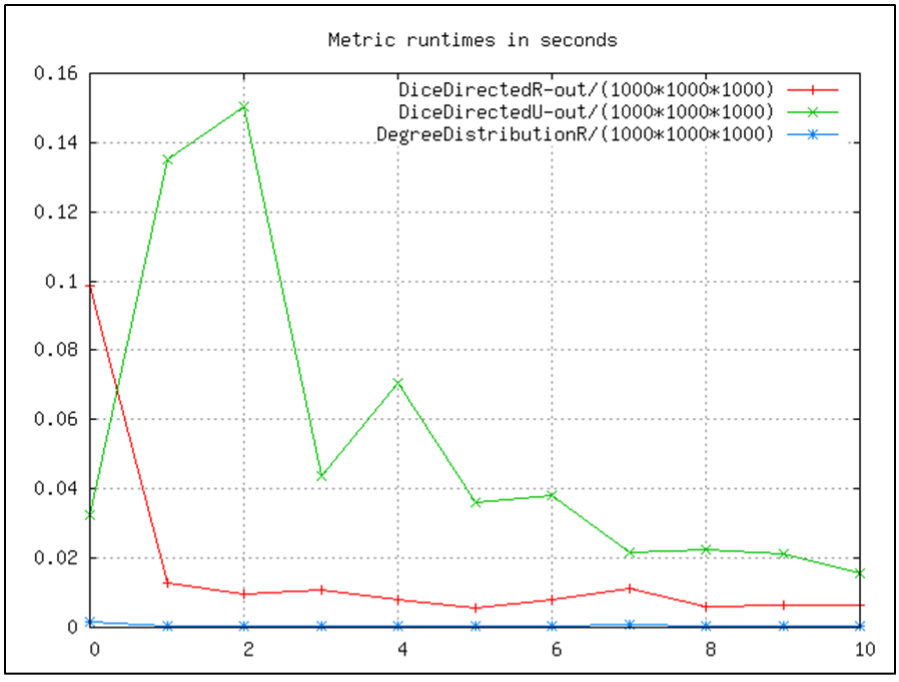


Figure 3.2 – Resulting plots from example 3.1.



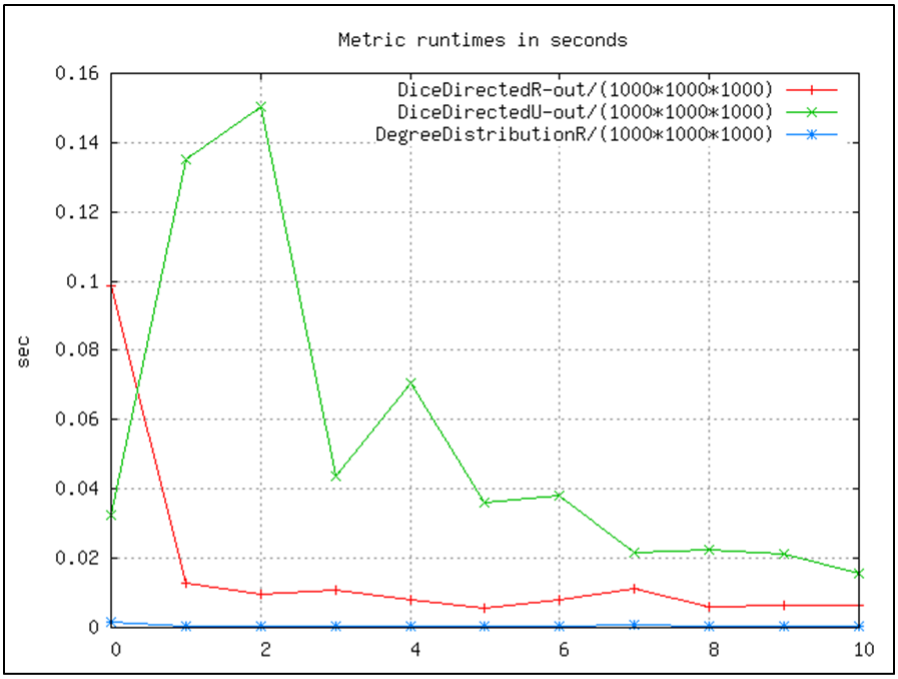

```
CUSTOM_PLOTS = METRICS
CUSTOM_METRICS_VALUES = metric_runtimes~*, general_runtimes~total
CUSTOM_METRICS_FILENAME = wildcard_example1
CUSTOM_METRICS_TITLE = Metric runtimes and total runtime
```

Figure 3.3 – Example plot with wildcards.



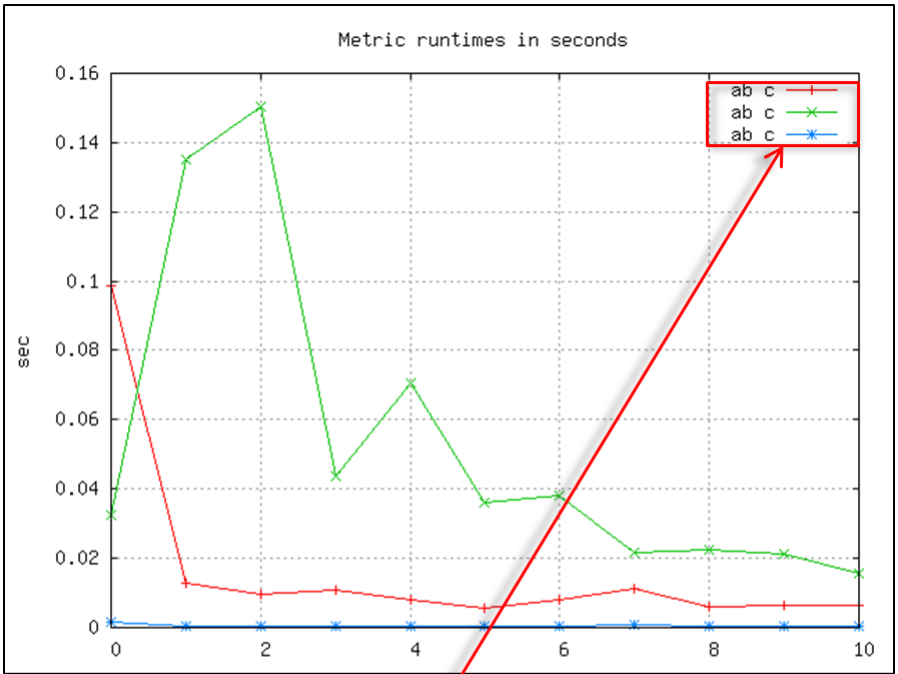

```
CUSTOM_PLOTS = METRICS
CUSTOM_METRICS_VALUES = :$metric_runtimes~*$/(1000*1000*1000)
CUSTOM_METRICS_FILENAME = math_expr_example1
CUSTOM_METRICS_TITLE = Metric runtimes in seconds
```

Figure 3.4 – Using mathematical expressions and wildcards to plot metric runtimes in seconds.



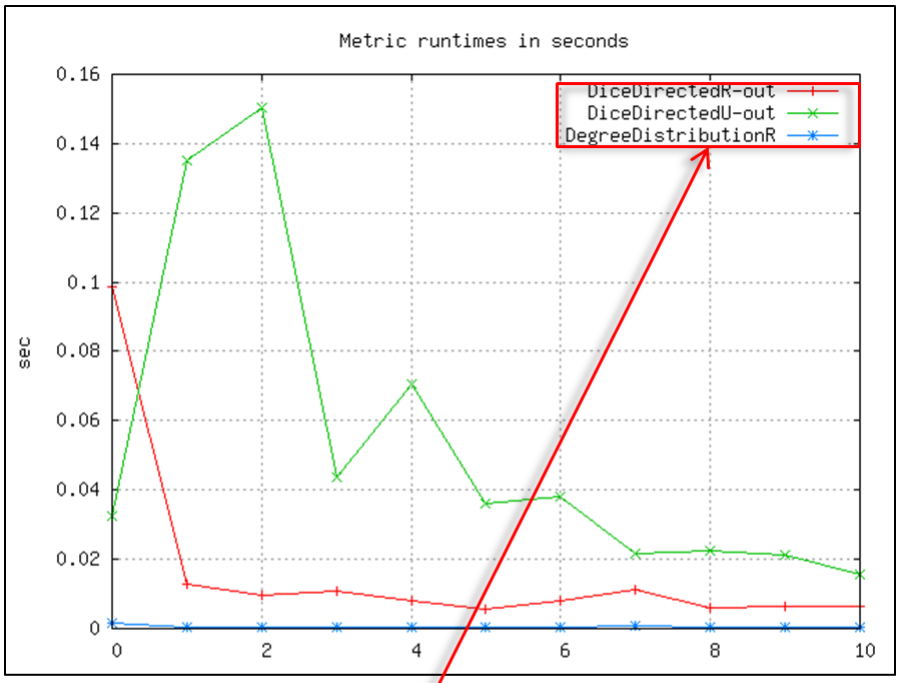

```
CUSTOM_PLOTS = METRICS
CUSTOM_METRICS_VALUES = :$metric_runtimes~*$/(1000*1000*1000)
CUSTOM_METRICS_FILENAME = math_expr_example1
CUSTOM_METRICS_TITLE = Metric runtimes in seconds
CUSTOM_METRICS_YLABEL = sec
```

Figure 3.5 – Adding a label to the y-axis.



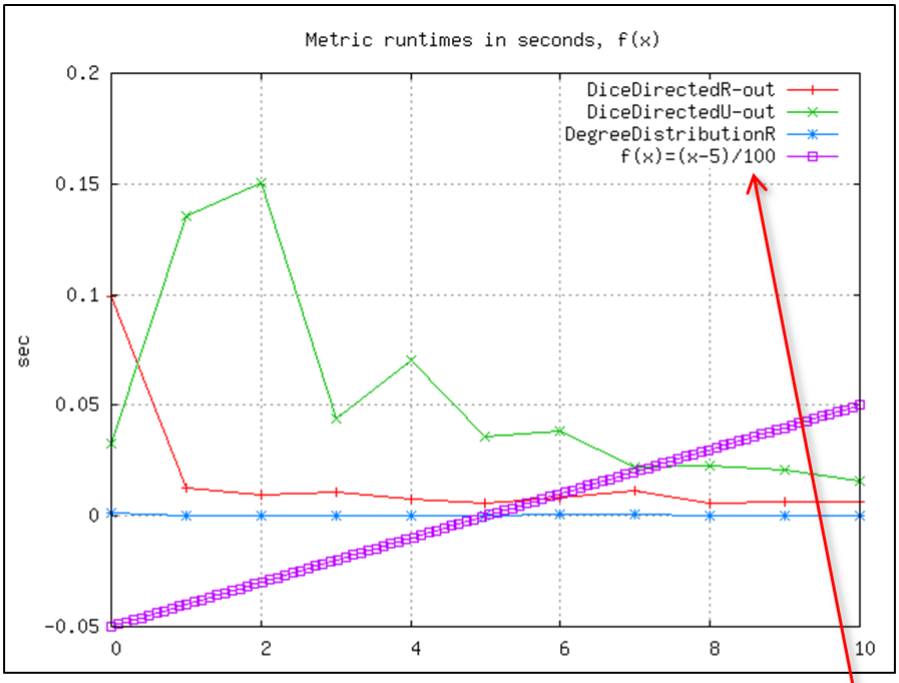

```
CUSTOM_PLOTS = METRICS
CUSTOM_METRICS_VALUES = "ab c":$metric_runtimes~*$/(1000*1000*1000)
CUSTOM_METRICS_FILENAME = math_expr_example1
CUSTOM_METRICS_TITLE = Metric runtimes in seconds
CUSTOM_METRICS_YLABEL = sec
```

Figure 3.6 – Assigning a name to the math. expression.



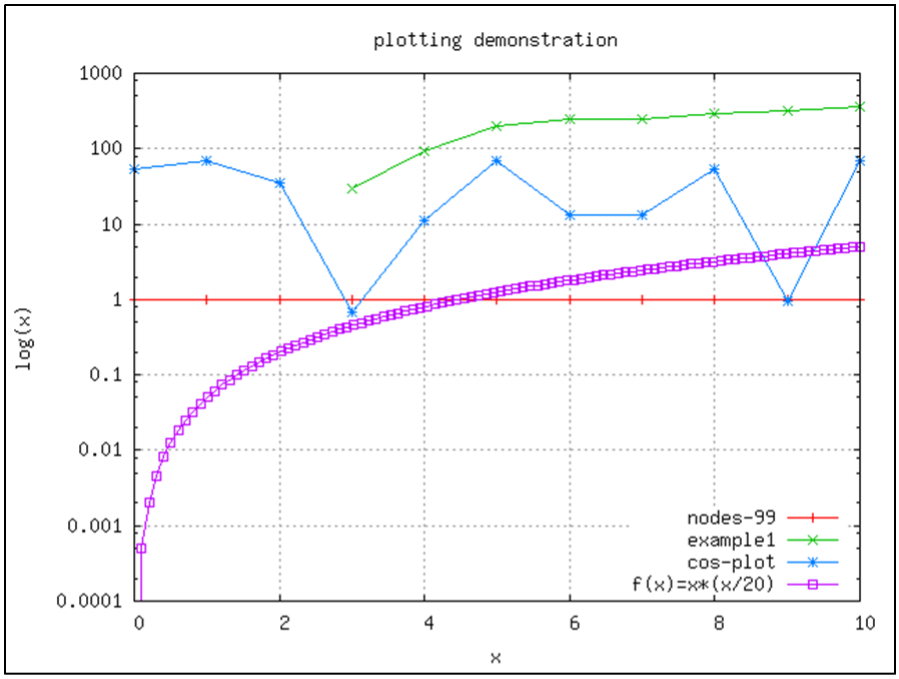

```
CUSTOM_PLOTS = METRICS
CUSTOM_METRICS_VALUES = $*$:$metric_runtimes~*$/(1000*1000*1000)
CUSTOM_METRICS_FILENAME = math_expr_example1
CUSTOM_METRICS_TITLE = Metric runtimes in seconds
CUSTOM_METRICS_YLABEL = sec
```

Figure 3.7 – Using wildcards in expression names.




```
CUSTOM_PLOTS = METRICS
CUSTOM_METRICS_VALUES = $*$:$metric_runtimes~*$/(1000*1000*1000), f(x)=(x-5)/100
CUSTOM_METRICS_FILENAME = math_expr_example1
CUSTOM_METRICS_TITLE = Metric runtimes in seconds, f(x)
CUSTOM_METRICS_YLABEL = sec
```

Figure 3.8 – Adding a mathematical function to the plot.




```
CUSTOM_PLOTS = M1
CUSTOM_M1_FILENAME = math_plot_example
CUSTOM_M1_TITLE = plotting demonstration
CUSTOM_M1_VALUES = "nodes-99":$statistics~nodes$-99,
example1:2*($statistics~nodes$+300)-$statistics~edges$, cos-
plot:35*cos($statistics~edges$)+34, f(x)=x*(x/20)
CUSTOM_M1_XLABEL = x
CUSTOM_M1_YLABEL = log(x)
CUSTOM_M1_LOGSCALE = y
CUSTOM_M1_KEY = right bottom
```

Figure 3.9 – Extended example demonstrating possible function definitions.