

Graph Analysis (GrAna)

Project Overview

Related Theses:

- Martin Rieß, *A Graph Editor for Algorithm Engineering*, September 2010 ([pdf](#))

The Graph Analysis (GrAna) project allows to examine a broad variety of structural properties of a graph (node count, edge count, etc.) as well as properties of the final drawing (area, edge crossings, etc.). An analysis can either be performed on a single graph or batch-like on large collections of graphs, in which case the results are written to a file. We support a variety of input formats, namely every format our formats service knows about.

- [Single Graphs Within the Editor](#)
- [Batch Executions](#)
 - [Textually](#)
 - [Eclipse Wizard](#)
- [Required Plugins](#)



A word of warning. Please scrutinize any results you get from GrAna. A lot of the existing analyses were written with certain graphs structures and use cases in mind. For instance, if the analysis does not account for self loops but your graph contains them, the results might be wrong.

Single Graphs Within the Editor

Within Eclipse we provide a *Graph Analysis* view, which can be found via Eclipse's *Windows->Show View->Others* dialog.

In the top right corner of this view you can find two buttons, one of which performs the analysis on the lastly selected diagram. The other button allows you to configure the analysis, i.e. select specific analyses to perform. The following screenshot shows an example. On the very left it can be seen that the analysis was configured to list the node and edge count, the area, and the number of edge crossings. According to the results, the graph contains 4 nodes and 5 edges. This is correct because the *hyperedge* between nodes N2, N3, and N4 is structurally composed out of two edges and just drawn as a single edge. Counting hyperedges could be a different analysis. Moreover, the area is reported as 25800 and 0 edge crossings are present.

Analysis	Result
Edge Count	5
Node Count	4

Analysis	Result
Area	215.0 (Width) 120.0 (Height) 25800.0 (Product)
Number of Edge Crossings	0 (Minimum) 0.0 (Average) 0 (Maximum) 0 (Sum) 0.0 (Onefold)

Batch Executions

A frequent use case is to analyze a large set of graphs with regards to some metric. For instance, you finished your awesome new algorithm to minimize the number of edge crossings in a graph and want to know how well it performs. The web offers large sets of graphs that already served as benchmarks in the past. GrAna allows batch analyses to be executed in two ways, textually and using an Eclipse Wizard.

Textually

We provide a concise DSL to specify the sets of graphs, the layout algorithm and options, and the analyses textually and store them in a *.grana* file. The DSL is based on Xtext, hence the editor comes with syntax highlighting, content assist, and formatting. An example specification can be found below.

You can specify a list of `Jobs`. A job is a self-contained unit of work. It specifies the previously mentioned data using the following keywords: `resources`, `layoutoptions`, `analyses`, and `output`. Layout options are specified by blocks that start with an arbitrary identifier followed by curly brackets. It is possible to specify multiple blocks of layout options. Each block results in a separate layout run allowing, for instance, to first execute a node placement algorithm and then an edge routing algorithm. For convenience it is possible to specify resources and output files globally at the beginning of the file and then use the `ref` keyword to reference them from a job.

Besides jobs, there are also `RangeJobs` and `CompareJobs`. A `RangeJob` can be used to analyze the effect of a specific layout option onto a specific metric. In the example below, the *thoroughness* layout option is registered using the `rangeoption` keyword. An integer range is specified resulting in all values between (inclusive) 1 and 50 being tested. While the analyses specified using the `analyses` keyword are only measured on the initial graph, the analysis specified using the `rangeanalysis` keyword will be measured for every tested value of the range layout option. Since analyses can be composed out of multiple components (e.g. the edge crossing analysis states the minimum, maximum, and average number of crossings per edge as well as the sum – four components), the `component` keyword tells GrAna which component to write to the output file. The `CompareJob` can apply two different layouts to a graph and compare the results (the graph is copied internally so the layouts do not influence each other). The two layouts are specified via two layout blocks as shown in the *avg_distance* example below. This kind of job can be used to analyze the effect of a layout on the individual nodes (e.g. the distance nodes are moved).

To execute GrAna based on a *.grana* file, right click the file and select *Execute Analysis Batch ...*

```
globalResources
  random "/Test/random/" filter ".*kgx"
  north_graphs "file:///D:/graphs/north" filter ".*graphml"
  secret "/Test/secret/" filter ".*json"

globalOutputs
  original_alg "/Test/results/original.csv"
  awesome_alg "file:///workspaces/eclps/Test/results/new.csv"
  thoroughness "/Test/results/thorough.csv"
  compare "/Test/results/compare.csv"

execute all

job original
  layoutBeforeAnalysis
  resources
    ref random
    ref north_graphs
    ref secret
  layoutoptions
    klay {
      de.cau.cs.kieler.algorithm: de.cau.cs.kieler.klay.layered
      de.cau.cs.kieler.klay.layered.crossMin: LAYER_SWEEP
    }
  analyses
    de.cau.cs.kieler.kiml.grana.nodeCount
    de.cau.cs.kieler.kiml.grana.edgeCrossings
  output ref original_alg

job my_awesome
  layoutBeforeAnalysis
  resources
    ref random
    ref north_graphs
    ref secret
  layoutoptions
    klay {
      de.cau.cs.kieler.algorithm: de.cau.cs.kieler.klay.layered
      de.cau.cs.kieler.klay.layered.crossMin: LAYER_SWEEP
      de.cau.cs.kieler.klay.layered.crossMin.awesome: true
    }
  analyses
    de.cau.cs.kieler.kiml.grana.nodeCount
    de.cau.cs.kieler.kiml.grana.edgeCrossings
  output ref awesome_alg

rangejob thoroughness
  resources
    ref random
  layoutoptions
    klay {
      de.cau.cs.kieler.algorithm: de.cau.cs.kieler.klay.layered
      de.cau.cs.kieler.klay.layered.crossMin: LAYER_SWEEP
```

```

    }
    analyses
    de.cau.cs.kieler.kiml.grana.nodeCount
    rangeoption
    de.cau.cs.kieler.klay.layered.thoroughness
    intrange 1 to 50
    // floatvalues 0.3, 0.4, 0.5
    rangeanalysis
    de.cau.cs.kieler.kiml.grana.edgeCrossings
    component 3
    output ref thoroughness

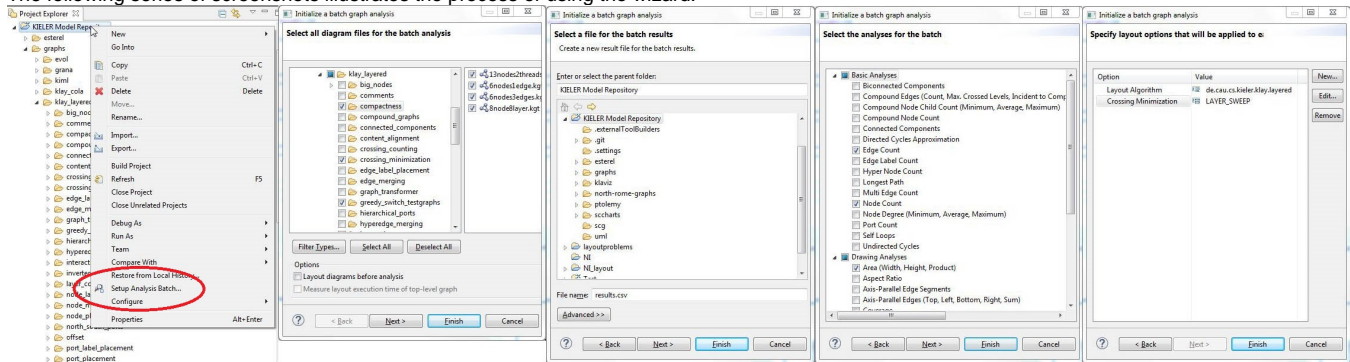
comparejob avg_distance
resources
    ref random
layoutoptions
    11 {
        algorithm: fixed
    }
    12 {
        algorithm: layered
    }
}
analyses
    de.cau.cs.kieler.grana.compare.averageDistance
output ref compare

```

Eclipse Wizard

Second, you can use an Eclipse *Wizard* to select one or more sets of graphs, specify and configure a layout algorithm, and select a set of analyses. Results are then written to a .csv file (or more precisely semicolon-separated-file).

The following series of screenshots illustrates the process of using the wizard.



Required Plugins

- Core:
 - de.cau.cs.kieler.grana
- Textual DSL:
 - de.cau.cs.kieler.config.text
 - de.cau.cs.kieler.config.text.ide
 - de.cau.cs.kieler.config.text.ui
 - de.cau.cs.kieler.grana.text
 - de.cau.cs.kieler.grana.text.ide
 - de.cau.cs.kieler.grana.text.ui