HowTo use this test environment for graph drawing

June 22, 2016

Contents

1 What is this

2 Computing a drawing for a graph and visualizing it

Enter 4 in the initial mode-selection-menu. After this you are asked for the graph and the algorithms via a dialog in your console. Then the drawings are computed, visualized in a Java swing JFrame and some measured quality properties, including the time needed to compute the drawing, are printed to the console for every algorithm.
Note: Every algorithm starts with the same initial vertex locations. These initial vertex locations are random values that may differ when mode 4 is called again with the same graph.

3 Computing a series of drawings for different algorithms and several graphs

3.1 Doing the computations

Enter 5 in the initial mode-selection-menu. This is doing a series of tests for one (or more) graph set(s). The informations which graphs, which algorithms and how often one graph is to be drawn with the same algorithm but different random initial vertex-locations can be passed via console dialog or with a config-file. For each drawing of one graph the time needed for computing it and several qualitative properties like the number of edge crossings is saved in a json-file at data/computation_data/<*starting time of the series*>.

3.1.1 Passing informations via a config-file

The user can define the settings for one series of tests in a config-json-file. The path to that file has to be passed as first argument when calling the main() method. Note: There are different kinds of config-json-files used in this Java-project. For which type is needed here see the example given in Figure 1.

3.1.2 Passing informations via console dialog

Iff no (valid) config-json-file of the Java-class TestConfig is passed as first parameter of the main() method, the user is asked for the information needed to do a series of graph drawing tests. These informations are the set(s) of graphs, the algorithms and the number of runs per graph and algorithm to be done.

3.1.3 Adding runs from OGDF (C++)

3.2 Evaluating

The results gotten from the execution of the series is collected in instances of the class ComputationData in the Java project. For every tested graph there is one such instance. For the correct interpretation of such an instance an instance of the class ComputationMetaData is needed, where the algorithms taken for this computation are specified. The ComputationMetaData object and the ComputationData objects (one per graph) are saved as json-files at:
<*pathToComputationData*>/<*starting time of the series*>/
By default <*pathToComputationData*> is set to data/computation_data/ relative to the location of this Java project.

By selecting 6 in the initial mode-selection-menu a csv-file from a previous computation can be calculated. The specification of the file to be created are asked via console dialog. Follow the steps of that dialog. The then created csv-file is in the same directory as the computation data that is evaluated by that file and there in the folder *csv*.

```
{
   mode:  5,
   selectedGraphs:  ["rome-graphs.json"],
   selectedAlgorithms:
       [
           "layoutAlgorithms.FRLayoutNoMapsNoFrame",
           "layoutAlgorithms.FRGrid",
           "layoutAlgorithms.wspd.FRWSPDb_b",
           "layoutAlgorithms.quadtree.FRQuadtree"
       ],
   externCPlusPlusAlgPaths:
       [
           "Rome-Lib/id_jr2_FMMMLayout",
           "Rome-Lib/id_jr2_FMMMLayout_single_Level",
           "Rome-Lib/id_jr2_GEMLayout",
           "Rome-Lib/id_jr2_SpringEmbedderFRExact"
       ],
   selectedSGrowth:  0,
   smallestS: 0.1,
   increasingValueOfS: 1,
   numberOfSValues = 1,
   selectedRecompFunctions:
       [
           {a:  5,
           b:  0,
           updateBarycenters:  true}
       ],
   selectedThetaGrowth = 0,
   smallestTheta = 1,
   increasingValueOfThetas = 1,
   numberOfThetaValues = 1,
   numberOfTests = 5,
   qualityCriterions:
       [
           "EDGE_LENGTH",
           "DISTANCE_VERTEX_VERTEX",
           "DISTANCE_VERTEX_NOT_INCIDENT_EDGE",
           "RATIO_GEOMETRIC_VERTEX_DISTANCE_TO_SHORTEST_PATH",
           "DEVIATION_FROM_THE_OPTIMAL_ANGLE"
       ],
   qualityCriterionAngle:  true,
   qualityCriterionCrossings:  true,
   maxNodesFRExact:  22500
}
```

Figure 1: config.rome.json: This is an example for a config-json-file of the Java-class TestConfig used for setting the configurations of a test series. The path to it can be passed as first parameter when the main method is called.

```
{
   "pathToGraphs":"data/graphs",
   "pathToGraphSpecifications":"data/graphs/specifications",
   "pathForIpeExport":"data/ipe-files",
   "pathToComputationData":"data/computation_data",
   "pathToCPlusPlusResults":"data/C++_results",
   "pathToSaveCSVFilesFromSplitTreeTest":"data/split-tree-test"
}
```

Figure 2: config.PATHS.json: This is an example for a config-json-file of the Java-class PathsConfig used for setting and saving different save paths. These are the default values.

Of course the data-files can also be evaluated by using extern programs.

4 Exporting a computed drawing to Ipe

After a graph was drawn by different algorithms in mode 4 (see Section 2 in this HowTo) any of those drawings can be exported to an ipe-File. After the drawings have been computed you are asked whether to export one of the drawings. You can export all of them but one at a time.

5 Changing data paths

The paths to the directories from where the graphs and the graph specifications are read and the paths to the directories to which the ipe-files, computation data, C++ computation data and split tree test csv-files are written, has some default values set in the source code of the program. They are all contained in the data-folder within the project folder.

These paths can be changed by modifying the file *config.PATHS.json* in the project-folder. It is created by calling the main()-method once. In Figure 2 an example for that file with the default values can be seen. All these paths are relative to the location of the project.

6 Adding graphs or graph sets

6.1 Creating a graph set of random graphs with this Java-project

Enter 7 in the initial mode-selection-menu. That way you can create new random graphs that are created by the EppsteinPowerLawGenerator from the JUNG-Package. It is an implementation of a graph generator that generates undirected graphs with power-law degree distributions and the Java-docs there refer also to "A Steady State Model for Graph Power Law" by David Eppstein and Joseph Wang. You can specify the number of graphs, the range of the number of vertices and the range of the factor number of edges per number of vertices. The EppsteinPowerLawGenerator is called with 1000 iterations but only the largest connected component of each created graph is taken and saved. The new graph-files are saved in the Rome-graph-format and a graph specification is automatically saved for them, too.

6.2 Adding an existing graph set

You can also add existing sets of graphs to the program. For that insert a folder containing the new graph-files to the path where the graph sets are saved and add a graph [set] specification as json-file

```
{
   "graphsAreStoredInDirectory":true,
   "directoryOrMethodCallParameter":"exampleGraphs",
   "numberOfGraphs":42,
   "shortDescription":"Set of example graphs"
}
```

Figure 3: exampleGraphSet.json: This is an example for a graph [set] specification json-file of the Java-class GraphReference. This a specification to a graph set with 42 graphs (each in one file) saved in the folder "exampleGraphs".

to the place where the graph specifications are saved. See for those paths Section 5. An example for such a specification file can be seen in Figure 3. Currently graphs in those formats can be read:

- GML-Format (Must have ending .gml)
- GraphML-Format (Must have ending .graphml)
- Graph-Format (Must have ending .graph)
- Rome-Graph-Format (Any ending different to the 3 endings above)

To read more graph formats add the desired graph readers and modify the source code to include them.