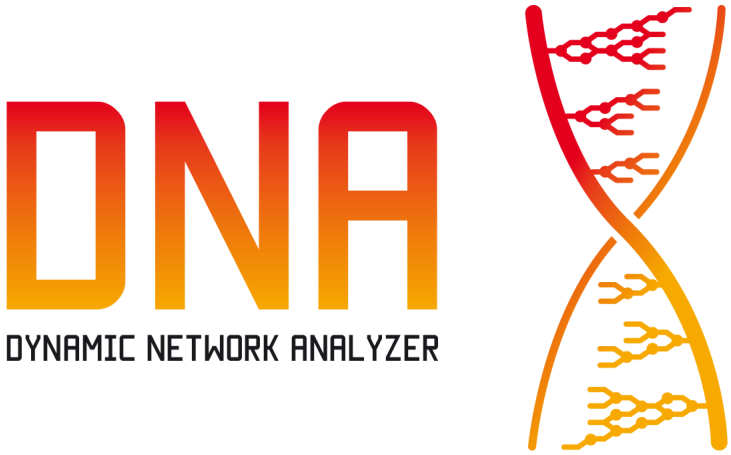

Graph Visualization
Instruction Manual

René Wilmes

20.05.2016

Dresden University of Technology

Contents

Chapter 1

Introduction

The graph visualization user interface of DNA - Dynamic Network Analyzer offers an intuitive and easy to use graph visualization tool with elaborate customization options. Figure 1.1 illustrates a sophisticated visualization of a graph modeling network activity. The following chapters will describe how to use gvis and how one may adapt it to specific cases.

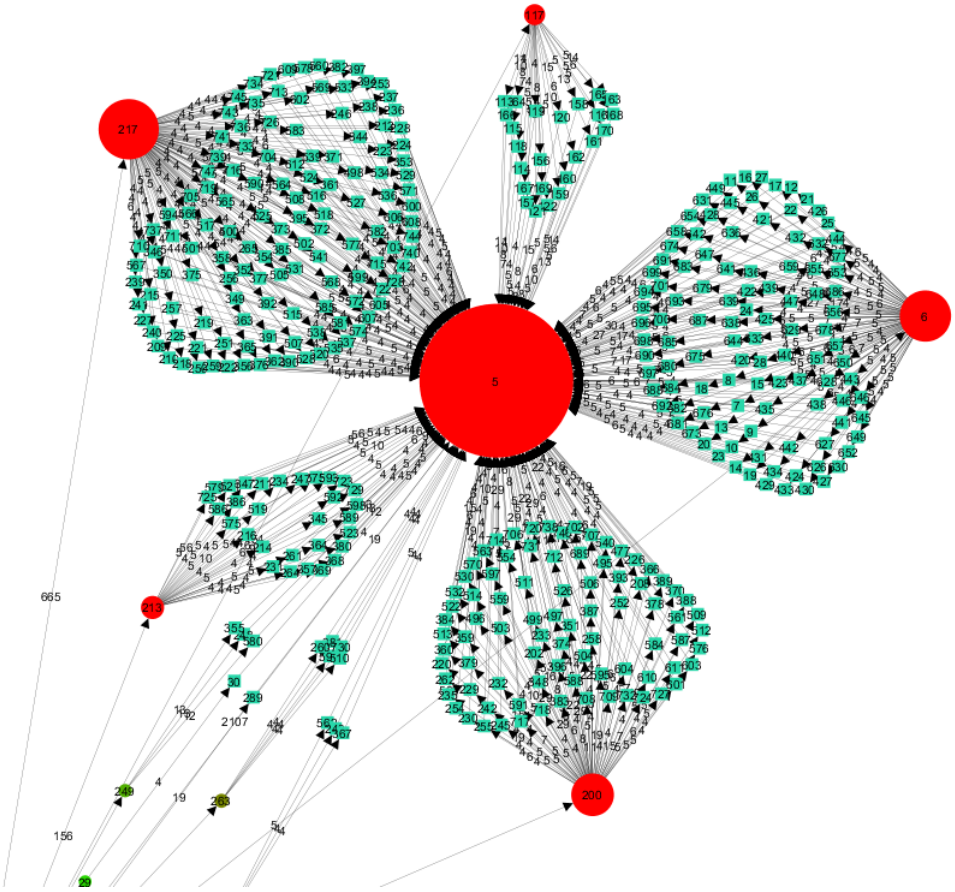

Figure 1.1 – Example of a DDoS-attack modeled as a graph.

Chapter 2

Graph Visualization

Gvis is designed to be intuitive and easy to use. It is implemented in java swing and uses the graphstream library [1] for the underlying graph visualization and layouting. Figure 2.1 shows an example gvis window.

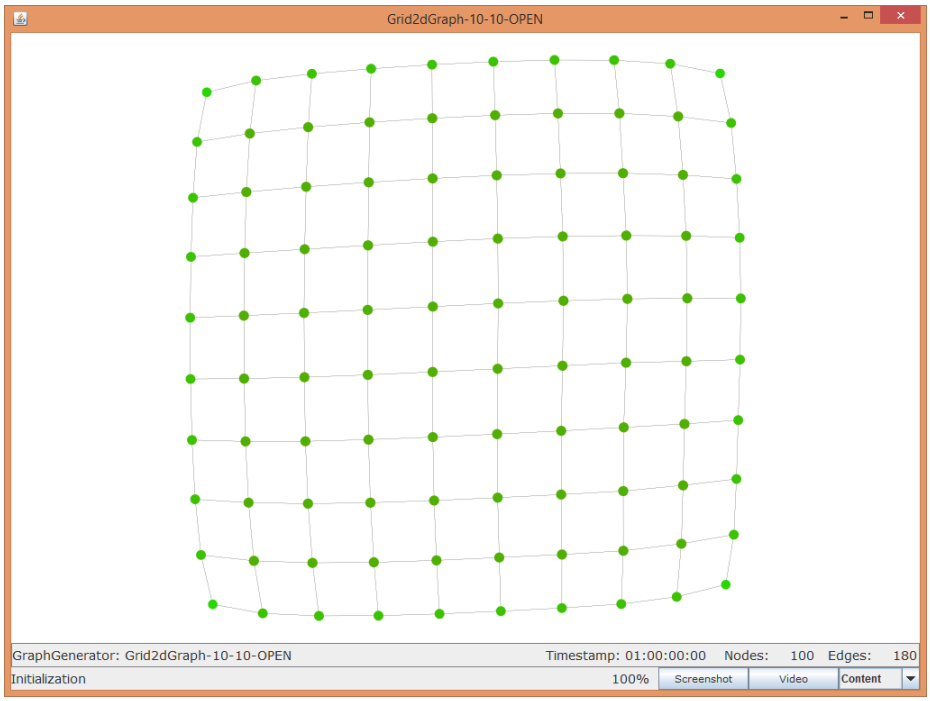

Figure 2.1 – Example of the graph visualization interface.

2.1 Interface

The interface is divided into two different areas, the graph area and the information area, see 2.2. The graph area displays the graph and allows some user interaction, for example dragging nodes, zooming with the mousewheel or moving in the graph with right mouse button pressed. The information area consists of the StatPanel, which shows basic information, and the TextPanel, which shows a status text and allows the user to capture screenshots or videos of the gvis window, see 2.3. The capture area selection shows what parts of the window will be captured with screenshots or videos. Selecting *graph* will only capture the graph area. *Content* will capture graph

and information area. *Full* will capture the entire window including its border. Note that recordings require the window to stay visible and at the same position.

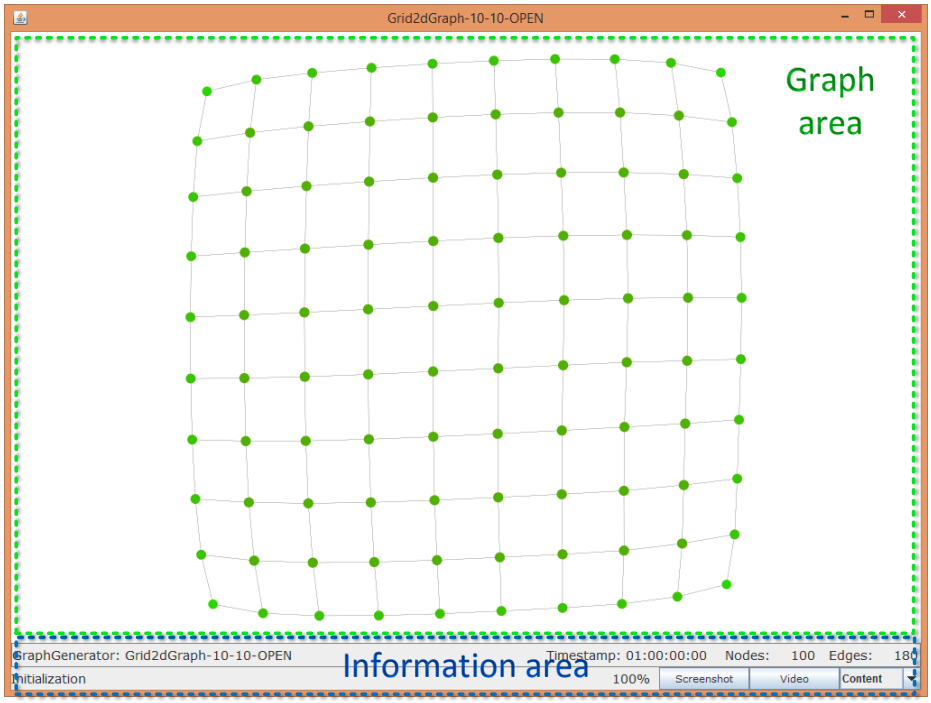

Figure 2.2 – Different interface areas.

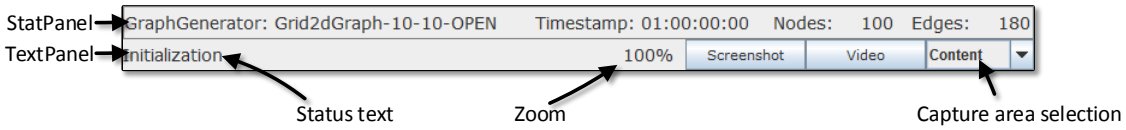

Figure 2.3 – The information area.

2.1.1 ToolTips

ToolTips are a special part of the graph visualization. They reveal additional information to the user and offer basic interactions with the underlying graph. Each node in the graph may have its own ToolTips, which can be shown and hidden by clicking the node with the right mouse-button. Figure 2.4 illustrates how the default ToolTips look like. Four ToolTips are visible:

1. NodeIdLabel: Shows the node id.
2. NodeDegreeLabel: Shows the nodes degree.
3. FreezeButton: May be clicked to toggle freeze-mode on the node. This will (mostly) prevent the node to be moved during layouting and dragging of adjacent nodes.

4. HighlightButton: May be clicked several times to highlight the node, increasing its size.

These ToolTips build the basic and default ToolTips to be used in gvis. However, the implementation of customized ToolTips is easy and will be discussed in section 3.1. How to enable ToolTips using the GraphVisConfig will be explained in section 3.3.2.

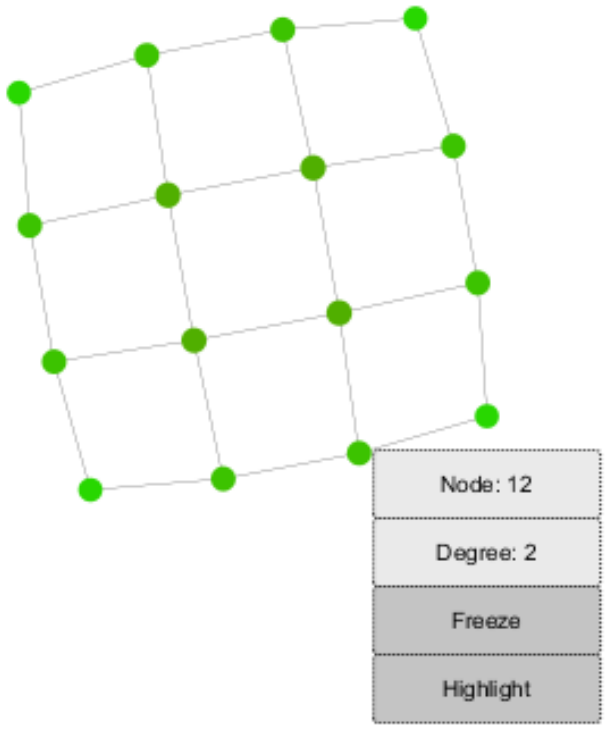

Figure 2.4 – Example of the default ToolTips.

2.2 Getting started

Using the graph visualization part of DNA is very simple. It can be enabled/disabled with:

```
GraphVisualization.enable();

GraphVisualization.disable();
```

When the visualization is enabled it is sufficient to generate a graph using DNA. The visualization window will be initialized automatically and all events, like node or edge additions, occuring in the graph will be visualized. Figure 2.5 shows a simple example resulting in a 2d grid-graph as shown in figure 2.1. For more details about graph generation in DNA check the generation documentation.

```
public static void main(String[] args) throws
        MetricNotApplicableException {
  // enable gvis
  GraphVisualization.enable();

  // init series
  GraphGenerator gg = new Grid2dGraph(GDS.undirected(), 10, 10,
      ClosedType.OPEN);
  BatchGenerator bg = new RandomBatch(5, 0, 10, 0);
  Series s = new Series(gg, bg, new Metric[0], "data/gvis-test/",
      "grid2dgraph");

  // generate
  SeriesData sd = s.generate(1, 0, false);
}
```

Figure 2.5 – Example of visualizing a 2d GridGraph.

Chapter 3

Configuration

The graph visualization user interface is designed to be intuitive and easy to use. For most use cases the default configuration will offer a convenient GUI. However, more sophisticated occasions may require to adapt the interface to specific needs. This is achieved by defining custom ToolTips, GraphStyleRules and custom configuration files via JSON [2]. The following sections will show how to work with the interface and create individual configurations.

3.1 ToolTips

The abstract ToolTip-class (*dna.visualization.graph.toolTips.ToolTip*) is a wrapper class for the GraphStream Sprite class. Sprites are objects inside the GraphStream graph that can be used to display information and data. ToolTip introduces several methods, which make the handling of sprites more enjoyable. There are two types of ToolTips currently implemented in DNA: Buttons and InfoLabels. They are both represented by an abstract class (*dna.visualization.graph.toolTips.button.Button* & *dna.visualization.graph.toolTips.infoLabel.InfoLabel*), which extend the basic ToolTip class. In order to implement own ToolTips one is encouraged to extend either Button or InfoLabel. Additionally, it is necessary to define a constructor as follows:

```
public constructor(Sprite s, String name, Node n, Parameter[] params)
```

It will be called on initialization with exactly this parameters. The Parameter[] array can be used to pass additional parameters. Beside the constructor each ToolTip implementation (in this example the class would be ”BananaToolTip”) has to have a method:

```
public static BananaToolTip getFromSprite(Sprite s) {
  return (BananaToolTip) ToolTip.getToolTipFromSprite(s);
}
```

For additional functionality, like mouse interactions, one can overwrite the *onLeftClick()* and *onRightClick()* methods. See the actual default ToolTip implementations for well documented examples. Also note that the *dna.visualization.graph.rules.GraphStyleUtils*-class offers a variety of utility methods, which can be used to set labels, change sizes and colors and more.

3.2 GraphStyleRules

The GraphStyleRules are rules designed to change the graphical representation of the given graph. All GraphStyleRules have to extend the abstract *GraphStyleRule* class (located in dna.visualization.graph.rules) and can override one or more of the following methods in order to dynamically behave and react to certain events:

- onNodeAddition(Node n, Weight w)
- onNodeRemoval(Node n)
- onNodeWeightChange(Node n, Weight wNew, Weight wOld)
- onEdgeAddition(Edge e, Weight w, Node n1, Node n2)
- onEdgeRemoval(Edge e, Node n1, Node n2)
- onEdgeWeightChange(Edge e, Weight wNew, Weight wOld)

The handed over *Node* and *Edge* objects are nodes and edges of the graphstream graph (not from DNA!). Therefore, one can make use of the huge styling possibilities included in graphstream. The utility class *GraphStyleUtils* contains several methods, e.g. *setShape()*, *setColor()* ..., which can be used to change the graphic presentation of the given nodes and edges. For example implementations check the included rules like *RandomNodeSize* or *RandomNodeColor*. In section 3.3.3 we will discuss how GraphStyleRules can be loaded and used in gvis.

3.3 GraphPanelConfig

The GraphPanelConfig is a configuration object for a GraphPanel. It contains the most general configurable aspects as well as additional configuration objects, namely a *WaitConfig*, *ToolTipsConfig*, *RulesConfig*, *ProjectionConfig* and a *CaptureConfig*. The general configuration structure is shown in figure 3.1.

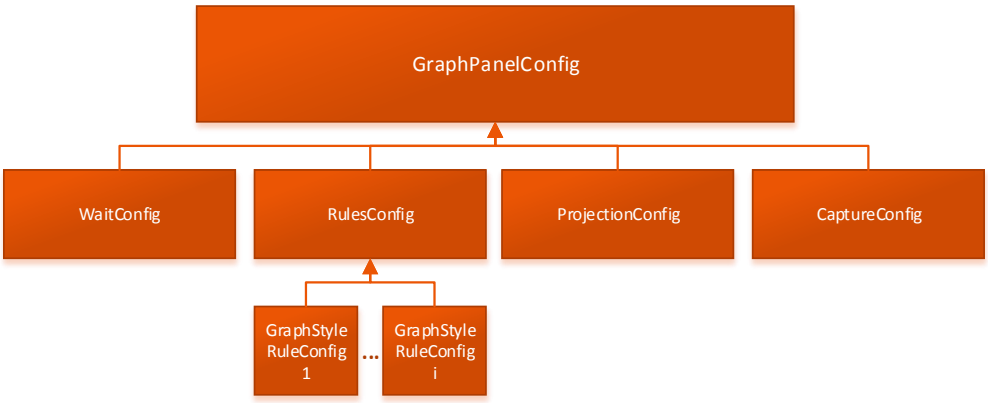

Figure 3.1 – Configuration Structure.

A default GraphPanelConfig will be loaded if no other config path is given explicitly prior to initialization. This may be achieved either by setting the respective property via

```
Config.overwrite("GRAPH_VIS_CONFIG_PATH", <config-path>);
```

or by calling the method:

```
GraphVisualization.setConfigPath(<config-path>);
```

All configurable values may be altered during runtime with getter- and setter-methods. In order to change values of subconfigs, e.g. the video record destination directory, one has to retrieve the CaptureConfig from the GraphPanelConfig, which is first retrieved from the GraphPanel directly. This specific case may look like this:

```
GraphPanel panel = GraphVisualization.getCurrentGraphPanel();
GraphPanelConfig config = panel.getGraphPanelConfig();
CaptureConfig ccfg = config.getCaptureConfig();
ccfg.setVideoDir("myvideos/");
```

However, specific values, which are only read and used at initialization, won't have any effect upon change. E.g. the window size of the graph panel is only read when the panel is being initialized. Any modification at a later point won't affect the GUI. All configurable values of the GraphPanelConfig and their descriptions are listed in 3.1.

3.3.1 WaitConfig

The WaitConfig object is used to configure special wait times used for smooth graph visualization. When adding multiple nodes and edges at once the layout algorithm may cause a very shaking and unstable view of the graph. Therefore, artificial wait times after each event are used to guarantee a smoother experience. Table 3.2 shows all configurable values.

3.3.2 ToolTipsConfig

In order to define which ToolTips will be displayed and how they are ordered and initialized one can use ToolTipsConfigs. The default ToolTipsConfig looks as follows:

```
"ToolTipsConfig": {
  "Enabled": true,
  "dna.visualization.graph.toolTips.infoLabel.NodeDegreeLabel": {
    "Name": "Degree",
    "Index": 1
  },
  "dna.visualization.graph.toolTips.infoLabel.NodeIdLabel": {
    "Name": "Node",
    "Index": 0
  },
```

Table 3.1 – Configuration values of GraphPanelConfig.

Value	Type	Description
Width	Integer	Width of the visualization window.
Height	Integer	Height of the visualization window.
Fullscreen	Boolean	If true the visualization window will be maximized on initialization.
StatPanelEnabled	Boolean	If true the visualization window will contain the StatPanel.
TextPanelEnabled	Boolean	If true the visualization window will contain the TextPanel.
TimetampFormat	String	Sets the used timestamp format.
RenderHQ	Boolean	Enables high-quality rendering.
RenderAA	Boolean	Enables anti-aliasing.
ZoomSpeed	Double	Zoom speed factor.
ScrollSpeed	Double	Scroll speed factor.
RotationSpeed	Double	Rotation speed factor.
DirectedEdgeArrowsEnabled	Boolean	If true directed edges will be rendered with arrow-heads.
NodeSize	Double	Default size of nodes.
NodeColor	String	Default color of nodes. All CSS color-keywords are available [3].
EdgeSize	Double	Default size of edges.
Layouter	Enum	Layouter used to layout the graph. Possible values: none, auto, linlog.
AutoLayoutForce	Double	Force of the auto-layouter.
WaitConfig	JSON-Object	A wait config. See section 3.3.1.
RulesConfig	JSON-Object	A rules config. See section 3.3.3.
ProjectionConfig	JSON-Object	A Projection config. See section 3.3.4.
CaptureConfig	JSON-Object	A capture config. See section 3.3.5.

```
    "dna.visualization.graph.toolTips.button.FreezeButton": {
      "Name": "FreezeButton",
      "Index": 2
    },
    "dna.visualization.graph.toolTips.button.HighlightButton": {
      "Name": "HighlightButton",
      "Growth": 15.0,
      "Index": 3
    }
  }
```

The ToolTipsConfig only has one static config field, which is *Enabled* and can be set true or false in order to enable/disable ToolTips. Additionally, all used ToolTips can be defined using their respective class-path as a key. The contained *Name* field will be used to uniquely identify the ToolTips. The *Index* field is used to set the order in which ToolTips will be shown in the visualization. All additional defined values (e.g. *Growth* in the HighlightButton definition) will be handed over to the ToolTip constructor as

Table 3.2 – Configuration values of WaitConfig.

Value	Type	Description
Enabeld	Boolean	If true wait times are enabled.
EdgeAddition	Long	Wait time after edge additions in ms.
EdgeRemoval	Long	Wait time after edge removals in ms.
EdgeWeightChange	Long	Wait time after edge weight changes in ms.
NodeAddition	Long	Wait time after node additions in ms.
NodeRemoval	Long	Wait time after node removals in ms.
NodeWeightChange	Long	Wait time after node weight changes in ms.

parameters, see section 3.1 for more details about the implementation.

3.3.3 RulesConfig

The graph visualization allows to define so called *GraphStyleRules* to modify the graph presentation in several ways. See section 3.2 for more details. In this section we focus on how to enable and configure said rules using the configuration. A RulesConfig holds multiple configurable *GraphStyleRuleConfis*, each representing a single GraphStyleRule. Each GraphStyleRuleConfig will be identified by a key pointing to the rules class-path. Additionally, there are three shared configurable values for each rule: the *name*, the *enabled*-flag and the *hidden*-flag. Furthermore, one may add additional parameters, which will be used by the respective rules. Note: The way these parameters will be treated is up to the given rule. See the following example 3.2 for an actual RulesConfig definition. The first rule contains all possible configurable values and one additional parameter: *Growth*. The NodeSizeByDegree class increases/decreases each nodes size by the growth-value whenever their degree is being incremented or decremented respectively. The second rule does not contain any values at all. However it will still be enabled.

```
"RulesConfig": {
  "dna.visualization.graph.rules.nodes.NodeSizeByDegree": {
    "Name": "defaultNodeSizeByDegree",
    "Growth": 0.3,
    "Enabled": true,
    "Hidden": false,
  },
  "dna.visualization.graph.rules.nodes.RandomNodeColor": {},
}
```

Figure 3.2 – Example RuleConfig definition.

3.3.4 ProjectionConfig

Since the graphstream library does not support three-dimensional coordinates, two simple projection mechanisms have been implemented to simulate 3D behaviour. In

Table 3.3 – Configuration values of a GraphStyleRuleConfig.

Value	Type	Description
Name	String	Name of the rule. (If not set *key* will be used as name.)
Enabled	Boolean	If set the rule is enabled. (If not set will be assumed as *true*.)
Hidden	Boolean	If set the rule is hidden from the interface. (If not set will be assumed as *false*.)
Parameter_0	Parameter	First parameter of the respective rule.
Parameter_1	Parameter	Second parameter of the respective rule.
...	...	...
Parameter_i	Parameter	i-th parameter of the respective rule.

this mode the node weights are assumed as 3D coordinates. For all configurable values see table 3.4.

Vanishing Point Projection

This projection mode uses a defined vanishing point and virtually *shifts* all nodes towards the vanishing point. How far the nodes will be shifted depends on the respective z-coordinate. The further a point is away from the screen-plain ($z = 0$) the further it will be shifted. The vanishing point coordinates, a scaling factor and logarithmic scaling can be configured in the ProjectionConfig. Note: If 3d-projection is enabled but vanishing point mode is disabled ortographic projection will be used instead.

Ortographic Projection

The ortographic projection uses a transformation matrix to project the three-dimensional nodes onto the two-dimensional screen-plain, see figure 3.3. The used matrix and a offset may be defined in the ProjectionConfig. Note: This mode is only used if the vanishing point projection is disabled.

$$\begin{pmatrix} n_x^{'} \\ n_y^{'} \end{pmatrix} = \begin{pmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \end{pmatrix} \cdot \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} + \begin{pmatrix} offset_x \\ offset_y \end{pmatrix}$$

Figure 3.3 – Ortographic projection from node n to two-dimensional node $n^{'}$.

3.3.5 CaptureConfig

The capture config defines how screenshot and video captures will behave. This includes basic options like destination directory as well as more detailed options, e.g.

Table 3.4 – Configuration values of a ProjectionConfig.

Value	Type	Description
Enabled	Boolean	If true 3d projection will be used.
UseVanishingPoint	Boolean	If true vanishing point will be used
VanishingPointLogScale	Boolean	If true the projection towards the VP will be scaled logarithmic.
VanishingPoint_X	Double	x-coordinate of the VP.
VanishingPoint_Y	Double	y-coordinate of the VP.
VanishingPoint_Z	Double	z-coordinate of the VP.
VanishingPointScaling	Double	Weight of the projection scaling.
ScalingMatrixS0_X	Double	x_0 value of the scaling matrix. For ortographic use only.
ScalingMatrixS0_Y	Double	y_0 value of the scaling matrix. For ortographic use only.
ScalingMatrixS0_Z	Double	z_0 value of the scaling matrix. For ortographic use only.
ScalingMatrixS1_X	Double	x_1 value of the scaling matrix. For ortographic use only.
ScalingMatrixS1_Y	Double	y_1 value of the scaling matrix. For ortographic use only.
ScalingMatrixS1_Z	Double	z_1 value of the scaling matrix. For ortographic use only.
OffsetVector_X	Double	x-value of the offset vector. For ortographic use only.
OffsetVector_Y	Double	y-value of the offset vector. For ortographic use only.

the frames-per-second included in the video file. For all available configuration values see table 3.5.

3.4 Default-Configuration file

The default configuration file located in ”config/gvis/gvis default.cfg” holds default values for each type of object and all possible properties. It represents the ultimate structure a configuration file can assume. When creating manual configuration files missing values will be replaced with the respective values from the default configuration, thus allowing to craft own configurations easily and with little effort. Note that any changes on the default file can prevent the interface from running, for example removing values that are needed as default references.

3.5 Building own configuration files

When creating an own configuration file it is recommended to take the default config structure and fill it with all desired values and settings. The following example 3.4

Table 3.5 – Configuration values of CaptureConfig.

Value	Type	Description
RecordArea	Enum	Default area to be recorded. Possible values: content, graph, full
VideoDir	String	Destination directory for captured videos.
VideoSuffix	String	Suffix of video files.
VideoFilename	String	Filename of captured video files. If set as ”null” names will be derived from the graph-panel name and (real-time) timestamp.
VideoMaximumLength	Integer	Maximum length of videos in seconds.
VideoRecordFps	Integer	Frames per second recorded from the screen.
VideoFileFps	Integer	Frames per second of the resulting video file.
VideoAutoRecord	Boolean	If set video recording will automatically start upon initialization.
VideoUseCodec	Boolean	If set video will be encoded using the Techsmith codec [4]. Else uncompressed.
VideoBufferImagesOnFs	Boolean	If true images will be buffered on the filesystem.
ScreenshotDir	String	Destination directory for screenshots.
ScreenshotFormat	String	Screenshot format (jpeg, png, bmp, wbmp, gif).
ScreenshotStabilityThreshold	Double	Stability threshold of the layouter when taking screenshots waiting for stability.
ScreenshotStabilityTimeout	Long	Stability timeout for taking screenshots waiting for stability in ms.
ScreenshotForegroundDelay	Long	Artificial delay when taking screenshots to let the machine move the panel to foreground.

shows a minimal configuration with the random node sizes and colors. Additionally the size is set to 640x480 and the stat- and text-panels are disabled. See figure 3.5 for the resulting GraphPanel.

```
{
  "GraphPanelConfig": {
    "Width": 640,
    "Height": 480,

    "StatPanelEnabled": false,
    "TextPanelEnabled": false,

    "RulesConfig": {
      "dna.visualization.graph.rules.nodes.RandomNodeSize": {},
      "dna.visualization.graph.rules.nodes.RandomNodeColor": {},
    }
  }
}
```

Figure 3.4 – Example of a small GraphPanelConfig.

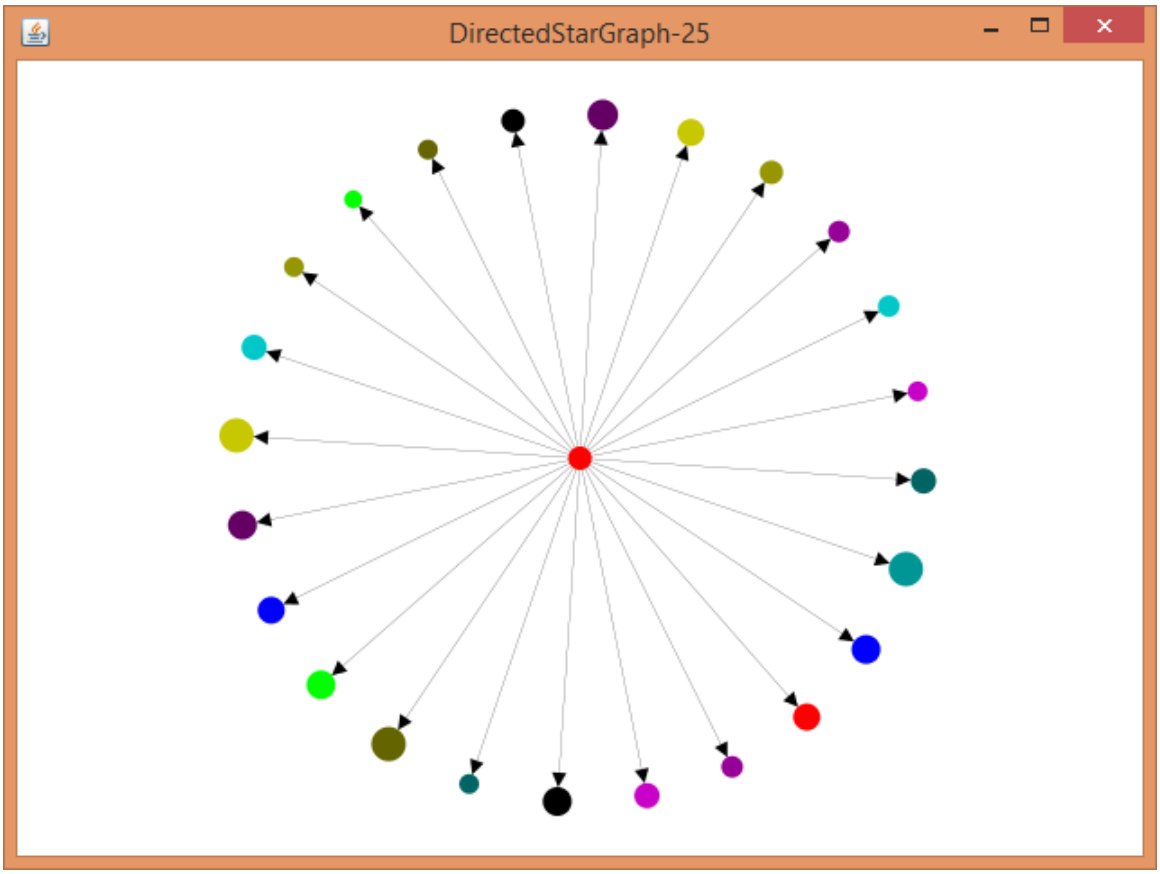

Figure 3.5 – Example visualization of the GraphPanelConfig from 3.4.

Bibliography

[1] Dutot, Antoine and Guinand, Frédéric and Olivier, Damien and Pigné, Yoann, *Graphstream: A tool for bridging the gap between complex systems and dynamic graphs*, ECCS 2007.

[2] Tim Bray, *The javascript object notation (json) data interchange format*, 2014. `http://rfc7159.net/rfc7159`.

[3] W3schools, *CSS Colors*, `http://www.w3schools.com/css/css_colors.asp`.

[4] TechSmith, *TSCC-codec*, `https://www.techsmith.com/codecs.html`.