Himmeli v3.0 user's guide

Ville-Petteri Mäkinen
Laboratory of Computational Engineering
Helsinki University of Technology

June 29, 2007

Contents

1 Getting started

Automatic graph drawing is perhaps the most widely applicable visualization aid that the advent of computers has brought to the scientific community. For a researcher, a clear picture of a complicated set of relationships helps to identify interesting features or to understand the large-scale nature of the problem at hand. The program Himmeli was developed to meet this need or, more specifically, to produce two-dimensional representations of weighted, directed and non-planar graphs. It is based on a force-directed algorithm that is capable of making acceptable vertex layouts of almost every type of network.

Himmeli has no graphical user interface[1] but creates a PostScript file of the graph instead. You can either print the file directly or open it with a viewer software such

[1] The installation of Himmeli is similar to that of CraneFoot software, see CraneFoot user's guide for details.

as GhostView. In addition, it is possible to convert the EPS-file[2] to the portable document format (PDF); in most Linux platforms the command `ps2pdf` works well for the fixed paper sizes. Another useful tool is `eps2pdf` by Wouter Kager and many image processing software (e.g. Gimp) accept EPS, too. If you are planning on using Himmeli for online graph drawing, the ImageMagick graphics library can be used for conversions from EPS to other image formats.

Himmeli requires two input files to work. The first, referred to as the *configuration file*, contains the instructions that specify where to look for the graph data and how to visualize it. Furthermore, the name of the configuration file is the only command line argument that Himmeli accepts, everything else is defined by the instructions. **If you call Himmeli without any command line arguments, a concise listing of all features is printed on the screen.**

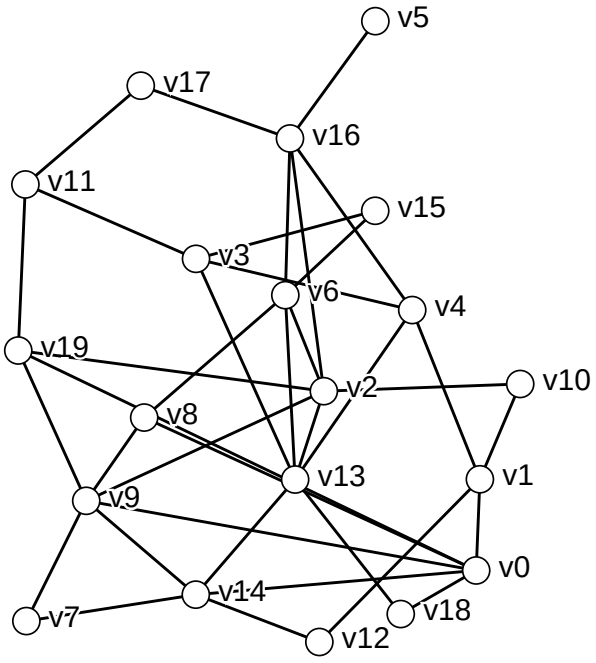

Figure 1: An undecorated drawing of the largest component of a non-planar random graph.

The second file, referred to as the *edge file*, contains the edges of the graph in tabulated ASCII text format. The first non-empty line of the file must contain the headings of the variable columns and the subsequent lines specify the graph topology. For each edge, you have to list at least the names of the endpoints on a single

[2] encapsulated PostScript

line in order for the edge to be registered by Himmeli. For instance, consider the first few rows and columns of the listing for the graph in Figure 1, written as

edges.txt:

```
HEAD    TAIL    WEIGHT  ..
v0      v1      0.100
v0      v8      0.100
v0      v9      0.100
v0      v14     0.100
v0      v18     0.100
v0      v19     0.100
v1      v4      0.200
v1      v10     0.200
v1      v12     0.200
v2      v6      0.300
v2      v9      0.300
:
```

Note that the columns are separated by horizontal tabulator characters that are not explicitly displayed here. If you wish to use another character, use the `Delimiter` instruction in your configuration file, details are at the end of this guide. The above alone is not enough, however, since without additional knowledge you cannot tell which column contains which variable. Hence the configuration file is needed to convey the necessary information to Himmeli. To produce Figure 1, we need five instructions, formulated as

config.txt:

```
GraphName               myGraph
EdgeFile                edges.txt
EdgeHeadVariable        HEAD
EdgeTailVariable        TAIL
EdgeWeightVariable      WEIGHT .
```

The first instruction `GraphName` sets the common part of output file names. Himmeli does not assume that the graph is connected, and it will print each component on a separate page (or file if so instructed, see `FigureLimit` for details). Largest components are printed first as can be witnessed in the output file *myGraph.ps*. The next line tells the name of the edge file, and the remaining two instructions indicate the way the relevant columns should be used by Himmeli. If you do not have the weights, simply omit the last instruction and Himmeli will assign a uniform weight to each edge. Note that it is not necessary to use the same column headings as in

the example, you could use 'Source' and 'Sink' instead of 'TAIL' and 'HEAD' or any other. In addition, you do not have to put the columns in any specific order. Finally, to ease the burden of data setup, Himmeli detects the names of the vertices automatically from the edge file so you do not have to provide any other information about the graph to produce a basic layout.

Figure 1 could be improved, however, since at the moment the weights and directions of edges are not displayed. Also, it would be interesting to see the vertex strengths, that is, the sum of adjacent edge weights. You can tell Himmeli to decorate the drawing automatically by setting `DecorationMode` on in the configuration file. The colors, edge widths and vertex sizes are now determined by the weights and strengths (Figure 2). The visual cues are normalized for each graph so they are not suitable for comparing networks that have different strength distributions. To make matters easier, you can influence the use of weights by transformations and filtering, more on that later.

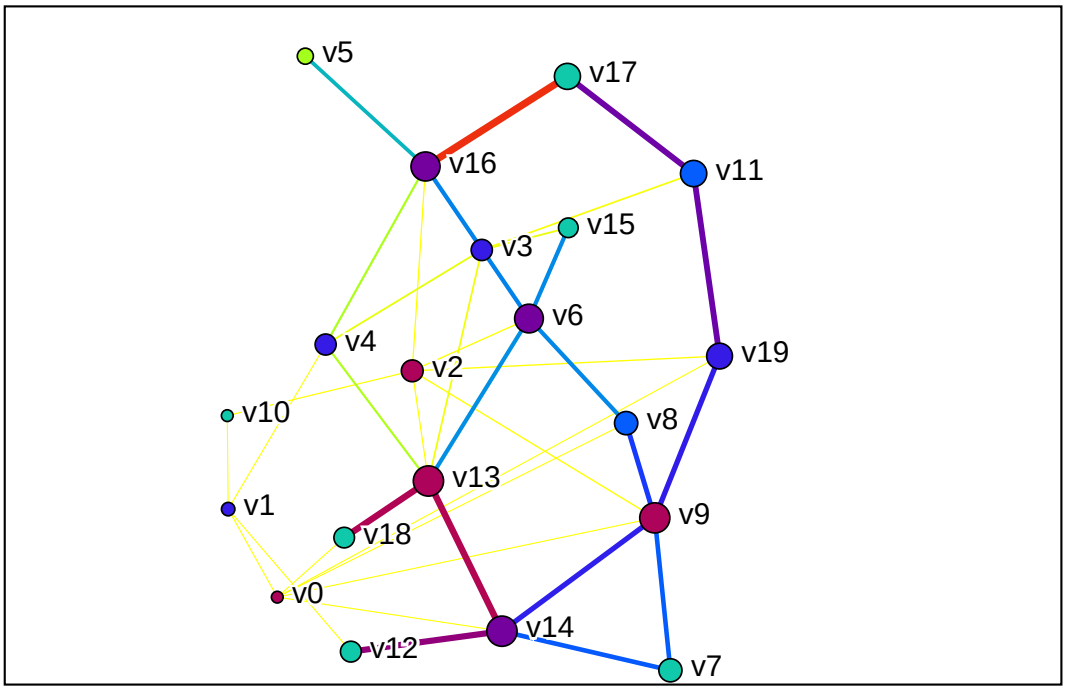

Figure 2: A decorated version of the weighted component in Figure 1.

When looking Figure 2 closely, you will notice that the layout is not identical with Figure 1. Graphs are high-dimensional objects so putting them on a 2D canvas is a brutal act. Furthermore, Himmeli's layout algorithm has a stochastic component that finds a minimum stress configuration – but not necessarily the global mini-

mum. Also, in this case the topology was slightly changed by automatic weight normalization.

2 Customizing outlook

Although Himmeli can automatically color the graph, it is often necessary to display specific information that requires custom visualization. For this reason, you can set the color, width and label for each individual edge in the edge file. Suppose you are interested in vertices v_{14} and v_{16} and wish to display only those edges that are adjacent to them. This can be accomplished by adding an extra column[3] in the edge file, written as

edges.txt:

```
HEAD    TAIL    WEIGHT  WIDTH ..
:
v11     v17     1.2     -1
v11     v19     1.2     -1
v12     v14     1.3
v13     v18     1.4     -1
v13     v14     1.4
v16     v17     1.7
:
```

Rather than explicitly setting the widths of the target edges, you can assign a negative width for the unwanted links instead. By doing this, Himmeli will still automatically decorate the unaffected edges, but skip the drawing of unwanted edges. It is important at this point to emphasize that the widths or any other visualization instructions have no effect on the vertex positioning algorithm, only the way the various objects are drawn will be altered.

To distinguish the two vertices from the rest, you may wish to use a different node symbol or color. For that, it is necessary to use a third input file, referred to as the *vertex file*. It has the same basic format as the edge file, that is, a header line of variable names followed by vertex records, each on a separate line. You need two columns: one for the vertex name and the other for the symbol code, formulated as

vertices.txt:

```
NAME    GROUP
```

[3]Himmeli also permits the use of auxiliary edge files, a topic that is discussed in a separate section.

```
v14     4
v16     4
:
```

The number 4 stands for the diamond shape, a complete list of shapes is depicted in Figure 3. As before, you must use the configuration file to pass this extra information to Himmeli, listed as

config.txt:

```
EdgeWidthVariable       WIDTH
VertexFile              vertices.txt
VertexNameVariable      NAME
VertexShapeVariable     GROUP .
```

The parameter `VertexNameVariable` indicates the column that contains the names of the target vertices. Notice that the names in the edge file are not all listed in the vertex file. In fact, Himmeli will detect only those vertices already present in the edge file but, as seen here, you do not have to include them all. Another important point is that you do not have to fix the order of edges or vertices in the data files, Himmeli can sort them automatically. The result is depicted in Figure 4.

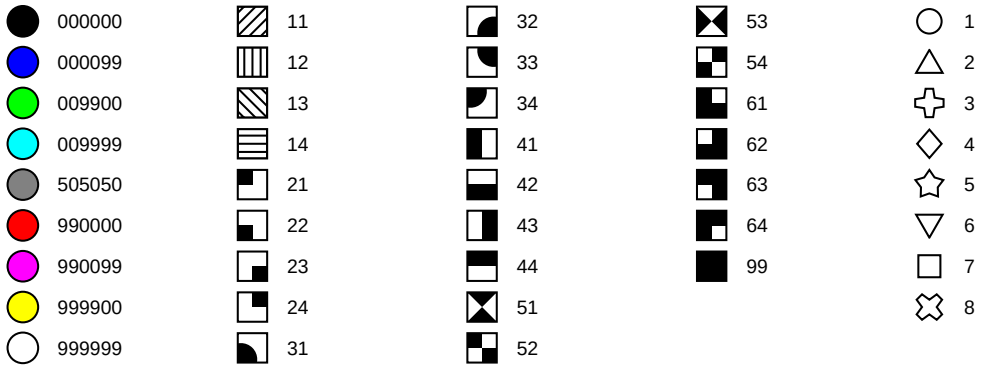

Figure 3: Node shapes, patterns and some color definitions.

3 Color and legend

Suppose that next you want to display the incoming and outgoing edges with different colors for the two vertices v_{14} and v_{16}. Again, you start by adding a column to the edge file and a corresponding instruction to the configuration file, written as

edges.txt:

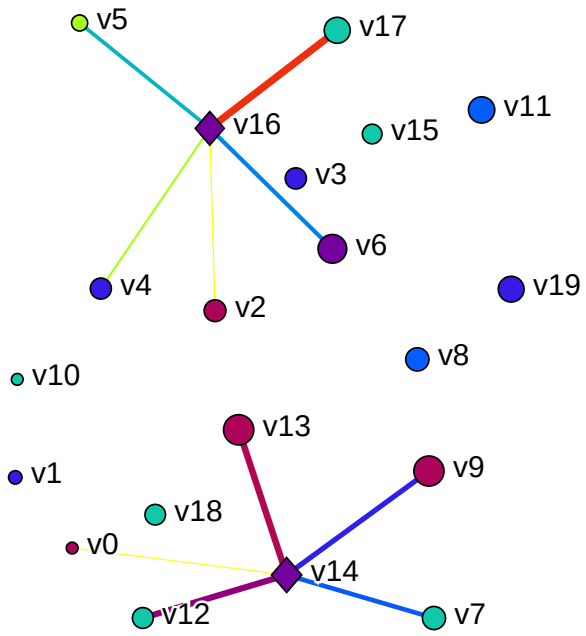

Figure 4: A customized version of the component in Figure 1.

```
HEAD    TAIL    WEIGHT  WIDTH   COLOR
v11     v17     1.2     -1
v11     v19     1.2     -1
v12     v14     1.3             000090
v13     v18     1.4     -1
v13     v14     1.4             000090
v16     v17     1.7             900000
:
```

and you can further emphasize the direction with an arrow head by enabling the `ArrowMode`. The human eye has three types of color receptors and for this reason colors can be defined as a combination of red, green and blue components. The values of the variable `COLOR` consist of three two digit blocks, each with the range from 00-99 indicating the strength of the primary color component. Here the aim was for a blue (000090) and red (900000). This is similar to the way colors are defined in HTML, for example, except that the decimal number system rather than the hexadecimal is used for clarity.

When adding shapes and colors to the graph presentation it is important to have a legend that describes the meanings of the various symbols. Himmeli collects the visualization instructions from the configuration file and prints the results on the

edge of each page, provided that `LegendMode` is on. Here it is instructive to add items that link the colors to edge directions and the diamond shape to the target vertices. To do this (and the previous enhancements), add the lines

config.txt:

```
ArrowMode           on
LegendMode          on
EdgeColorVariable   COLOR
EdgeColorInfo       incoming   900000
EdgeColorInfo       outgoing   000090
VertexShapeInfo     target     4 .
```

to the configuration file. The first value field of the “info” instructions sets the label and the second specifies the corresponding shape or color, as seen in Figure 5. The info parameters do not affect the graph drawing itself so you can add one even if you do not use that particular item in the current presentation.

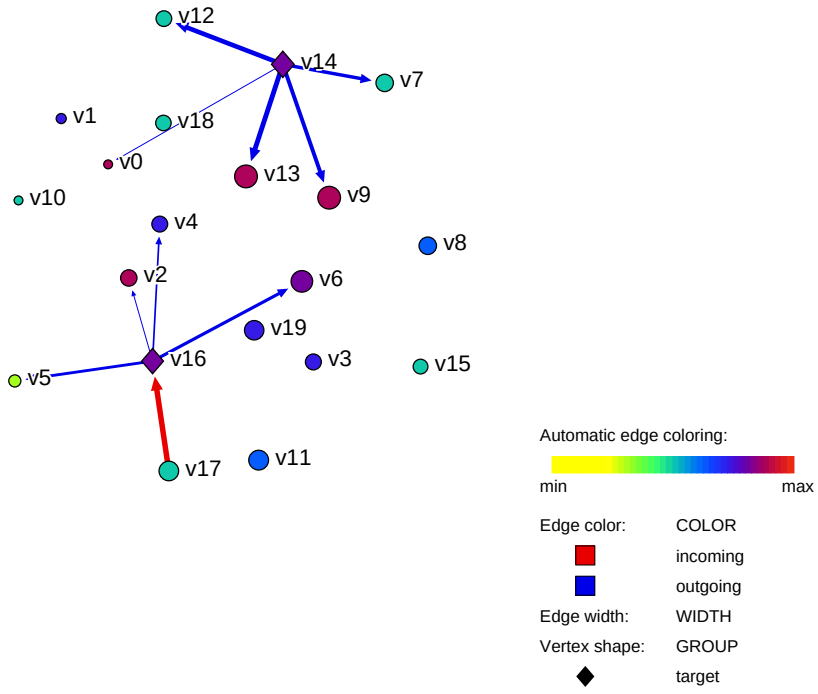

Figure 5: A customized version of the component in Figure 1 and the respective legend.

4 Layout reconstruction

Previously, the discussion was focused on the file formats and visual outlook of the drawings but an equally important issue is the quality of the vertex positioning. Himmeli employs a force-directed algorithm that assigns repulsive and attractive forces between vertices based on their mutual distances and connections. By simulating this artificial system of strings, Himmeli searches for a minimum energy state that hopefully has as few edge crossing and node overlaps as possible.

Positioning the vertices by simulation makes it difficult to determine when the graph has settled to an optimal configuration. The limiting factor in this process is time, since the rate at which large graphs converge to an acceptable layout is slow and the simulation itself is computationally heavy. For this reason, you can instruct Himmeli to stop after a fixed time quota by invoking the `TimeLimit` parameter.

Suppose you have a very large graph and are uncertain about the time it takes to produce an acceptable vertex layout. Most likely, you will not even know what that layout would look like. It is therefore important that you can continue a simulation if you think that the time quota was too short. In effect, this means that you want to pass the coordinates of the previous simulation run to the next so that Himmeli does not have to start from scratch on each occasion.

Auxiliary input files

By this time you have probably noticed that, in addition to the PS-file, Himmeli creates three text files that can be used to quickly reconstruct the layout. The accepted edges are stored to a new edge file, named according to `GraphName`. The file contains edge colors, widths and labels, as they appear in the figures. Similarly, the new vertex file has columns for the visual variables, but it also contains the node coordinates, degrees and strengths. The third text file is a new configuration file that instructs Himmeli to use the auxiliary edge and vertex files to import a node layout and visualize it accordingly.

To ensure that the graph topology remains unaffected, the updated version of the configuration file conserves some of the original instructions, but overrides those pertaining to visual cues and simulation time. By setting a positive `TimeLimit`, you can start a new simulation run that starts from the stored node layout. You can also use the auxiliary files as a template for customization, note particularly the way each variable can be fetched from a separate file if necessary.

Sometimes it is necessary to alter the visual outlook but to keep the positions. For

instance, if you wish to display the evolution of connections as a series of figures, it is essential that the vertices do not move between consecutive frames. Based on the previous discussion, this can be achieved by first exporting the node coordinates from a network that represents the entire series and then using this layout in every frame. By keeping the time quota at zero, Himmeli will not update the initial positions and therefore only the visual outlook of the edges and vertices is changed. Note, however, that if some of the coordinates are missing, Himmeli will use arbitrary values in their place.

5 Normalization and thresholding

Usually, the weights of a graph are not constrained. The node-positioning algorithm, however, works best if the edge weights are not much greater than one and not very close to zero (non-positive weights are not accepted). To tackle the problem, Himmeli has automatic normalization procedures that map the edge weights to the desired value range. There are altogether four options: you can set `EdgeWeightTransform` to fully automatic, linear or logarithmic scaling, or rank-based transformation.

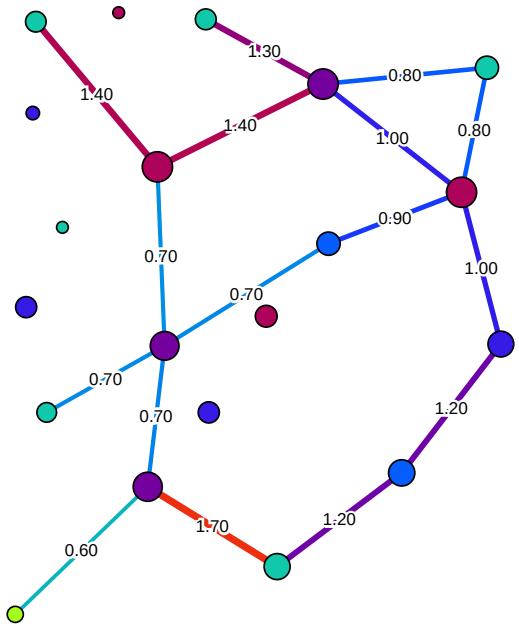

Figure 6: The component in Figure 1 after thresholding by the median weight of the random graph. Only those edges with a weight above the median are drawn.

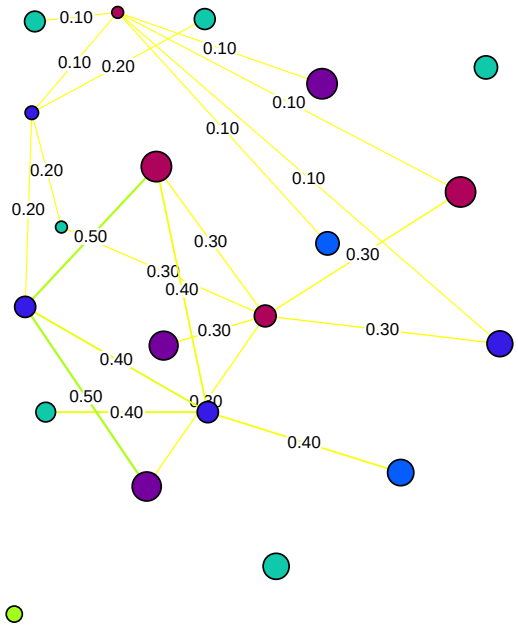

Figure 7: The component in Figure 1 after thresholding by the median weight of the random graph. Only those edges with a weight below the median are drawn.

Another typical need for a researcher is the thresholding of edge weights, that is, you either want to exclude or not draw some of the edges according to a constraint. For the former, you can use `EdgeWeightFilter` and to remove edges from the visualization step only – they are used when positioning the nodes – try the `EdgeWeightMask` instruction. Both features have options that let you threshold by weight quantile or absolute value (Figure 7 and Figure 6).

6 Configuration interface

File instructions

`EdgeFile` (required)
: Edge data file.

`GraphName`
: Base name for output files. The template is *<base>_<index>.eps* for graph components and *<base>.<type>.txt* for the reconstruction files.

`VertexFile`
: Vertex data file. You do not have to supply data for every vertex. Only those vertices that are also present in the edge file will be recognized.

Structural instructions

`EdgeHeadVariable` (required)
: The column of head vertices in the edge or other file.

`EdgeLabelVariable`
: The column of custom edge labels.

`EdgeTailVariable` (required)
: The column of tail vertices in the edge or other file.

`EdgeWeightFilter`
: To threshold edges by weight before node positioning, three values must be supplied: the type of filter (`abs/frac`) and two limits. For example, to include only the largest 10% edge fraction in the edge file, type `EdgeWeightFilter frac 0.9 1.0`.

`EdgeWeightMask`
: Same as `EdgeWeightFilter` except that edges are not excluded until after node positioning.

`EdgeWeightTransform`
: Transform edge weights to optimal range for node positioning. There are three types of transformations: linear scaling (`lin`), logarithmic scaling (`log`) and rank-based normalization (`rank`). In addition, `auto` is a combination of the three and `off` uses raw weights. The default is `auto`.

`EdgeWeightVariable`
: The column of numerical edge weights. Only positive values are accepted. If a weight variable is not supplied, each edge is given a uniform weight.

`VertexNameVariable`
: The column in the vertex or other file that contains the list of vertex names.

`VertexXVariable`
: The column of horizontal positions in the vertex or other file. The scale is normalized so that vertices should be at least a unit distance apart.

`VertexYVariable`
: The column of vertical positions in the vertex or other file. The scale is normalized so that vertices should be at least a unit distance apart.

Visualization instructions

`EdgeColorVariable`
: The column of color definitions in the edge file. Colors are determined by integer codes: 000000 for black, 505050 for gray, 999999 for white, 990000 for red, 009900 for green, 000099 for blue, or any other between 000000 and 999999.

`EdgeColorInfo` (multiple)
: Labels of specific edge colors to be depicted in the legend. Two value fields: label and code. Valid codes within $[000000, 999999]$.

`EdgeWidthVariable`
: The column of edge width multipliers. Non-negative values greater than 2 or smaller than 0.2 are truncated to the limits. A negative value will make the edge invisible.

`VertexColorInfo` (multiple)
: Labels for specific vertex colors. See `EdgeColorInfo`.

`VertexColorVariable`
: The color definitions in the vertex file, see `EdgeColorVariable` for usage.

`VertexLabelVariable`
: The column that contains new name labels in the vertex file. The new names will be used only in the figure, they will not be used to identify vertices.

`VertexPatternInfo` (multiple)
: Labels for specific patterns. Two value fields: code and label. Valid codes within $[1, 99]$, see Figure 3.

`VertexPatternVariable`
: Individual patterns. Valid codes within $[1, 99]$, see Figure 3.

`VertexShapeInfo` (multiple)
: Labels for specific vertex shapes. Usage is similar to `VertexPatternInfo`.

`VertexShapeVariable`
: The column of shape definitions in the vertex file. Shapes are determined by integer codes, see Figure 3 for details.

`VertexSizeVariable`
: Size multiplier for vertices.

Formatting and functional instructions

`ArrowMode`
: If set to `on`, each edge will be augmented with an arrow head. The default is `off`.

`BackgroundColor`
: Canvas background color, see `EdgeColorVariable` on representing the three rgb components. The default is 999999 (white).

`DecorationMode`
: If set to `on`, Himmeli will decorate the graph based on edge weights and vertex strengths. The color palette cycles from yellow (low weight or strength) through green and blue to bright red (high weight or strength). The default is `off`.

`Delimiter`
: The character that separates the columns in the edge and vertex files. Use `tab` for the horizontal tabulator and `ws` for white space. The default is `tab`.

`DistanceUnit`
: Set the standard distance between nodes on the canvas. This is not a zoom; it scales the coordinate system that is used when drawing the figures in such a way that the node symbols are not modified, only their mutual distances are changed. The default is 1.

`FigureLimit`
: The maximum allowed number of EPS image output files. Cannot be more than 9999, default is 0.

`FontSize`
: Set the font size for vertex labels. This has no effect on the legend. The default is 10pt.

`ForegroundColor`
: Line color for contours and text, see `EdgeColorVariable` on representing the three rgb components. The default is 000000 (black).

`IncrementMode`
: If set to `on`, Himmeli will start simulating from the layout that is supplied in the vertex file. The default is `off`.

`LabelMode`
: If set to `vertex`, the drawing is augmented with vertex names or user supplied labels. If set to `edge`, edge weights (or labels) are written on edge center points. If set to `on`, both previous options apply. The default is `vertex`.

`LegendMode`
: If set to `on` and there is visualization information available, a legend is drawn on each page on the main output file. The default is `off`.

`PageOrientation`
: Figure orientation, can be `landscape` (default) or `portrait`.

`PageSize`
: Keyword for page sizes for the output documents. Possible choices are 'letter', and 'a0' to 'a5' for the first value (main document). The second value refers to the component outputs (.eps files) with an additional option of automatic size 'auto'. The defaults are 'letter' and 'auto' for the first and second value, respectively.

`TimeLimit`
: The time quota in seconds for computing the node positions. If set to zero, no simulation is performed. The default is 10s.

`TreeMode`
: If set to `on`, Himmeli will use only the maximum spanning tree instead of the full graph. The default is `off`.

`VerboseMode`
: If set to `on`, Himmeli will print all runtime messages. The default is `on`.