Method 1 – naming the general style

lay out graph using? style automatic | circular | tree | hierarchical | symmetric | general

Method 2 – naming one or more aesthetic criteria

lay out graph (conforming to)? criteria? *criterion1, criterion2, criterion3…*

So:

1. lay out graph conforming to *criterion1, criterion2, criterion3…* or
2. lay out graph criteria *criterion1, criterion2, criterion3…* or just
3. lay out graph conforming to criteria criteria *criterion1, criterion2, criterion3…* or just

Criteria:

- (minimization of)? edge crossings
- planarity
- (maximization of minimum)? angle
- (minimization of)? Bands
- uniform? Flow
- symmetry
- even nodes distribution
- optimization of edges lengths
- similar edges lengths

Method 3 – naming the algorithm

lay out graph using? algorithm? *name((booleanProperty, property1 = value1, property2 = value2…))?*

Listing proeprties and setting their values is completely optional. Any number of properties can be set. If a property is ommited, a default value will be used. The properties don‘t have to be listed in any specific order. INT stands for integer numbers and FLOAT for real ones.

The following algorithms are available:

- radial tree(horizontal distance? = INT, vertical distance? = INT)
- level based tree(horizontal distance? = INT, vertical distance? = INT)
- compact tree(horizontal, invert, resize parents, level distance = INT, node distance = INT)

- node link tree(oritentation=left|right|up|down, spacing between? siblings = FLOAT, spacing between? subtrees = FLOAT, spacing between? depth? Levels = FLOAT, offset for? root node? = FLOAT)
- balloon tree(minimum radius = INT)
- hierarchical(reize parents, move parent, parent border, same layer spacing, adjacent layers spacing)
- circular(optimize crossings, distance = INT)
- concentric symmetric(automorphism = (v1, v2, v3))
  - The parameter is an automorphism in cyclic notation. Vertices should identified by their content or its index written after the letter 'v'
- spring(stretch = FLOAT, repulsion range = INT, force multiplier = FLOAT)
- Kamada-Kawai(disconnected? Distance multiplier = FLOAT,length factor = FLOAT, maximum iterations = INT)
- Fruchterman-Reingold(attraction multiplier = FLOAT, resolution multiplier = FLOAT, maximum iterations = INT)
- fast organic(force constant = FLOAT, minimal distance limit = FLOAT, initial temperature = FLOAT, maximum iterations = INT)
- organic(optimize edge crossings, edge crossing factor = FLOAT, optimize edge distance, edge distance factor = FLOAT,optimize border line, border line factor = FLOAT, optimize node distribution, node distribbution factor = FLOAT, fine tune, fine tuning radius = FLOAT, average node area = FLOAT, average scale factor = FLOAT)
- box(columns = INT)
- Tutte embedding(distance = INT)
- convex
- orthogonal (algorithms still in development)

Laying out subgraphs

It is also possible to specify how one or more subgraphs should be laid out, and not just compete graphs. Everything described for graphs can also be applied on subgraphs. Each subgraph can be laid out differently. A subgraph is defined by listing vertices that it contains, with the vertex being identified by its index preceded by the later 'v' or its content. If more subgraphs are listed, they are separated by a semicolon. Key word 'others'is used to specify how all vertices not previously listed belonging to the graph in question should be laid out. Character + specifies that something can be written 1 or more times.

lay out

```
        (subgraph containing? vertices v1, v2, v3… method1|method2|method3)+
        others method1|method2|method3
```

Example:

```
        lay out
         subgraph v3,v4 using style automatic;
         subgraph v0,v1,v2 conforming to uniform flow;
         others using algorithm Kamada-Kawai
```