In[1]:=
```
(*
 * This file contains examples for the functionality of the package Calligraphs.wl
 * Authors: Georg Grasegger, Boulos El Hilany, Niels Lubbes
 *)
```

In[2]:=
```
(*
 * WARNING: Most of the functions do not check whether the input is suitable.
 * If not, the output is not suitable either.
 * If you are uncertain what a suitable input for some FUNCTION is, check: ?FUNCTION
 *)
```

Setup

In[3]:=
```
SetDirectory[NotebookDirectory[]];
```

In[4]:=
```
(*
 * The packages Calligraphs.wl and LamanGraphs.wl need to be loaded.
 * Calligraphs.wl can be downloaded from https://doi.org/10.5281/zenodo.6421148
 * LamanGraphs.wl should be downloaded from [1]
 * https://doi.org/10.5281/zenodo.1245506
 * and stored in the same folder containing this file.
 * Run this cell to load the two packages.
 *)
Get["LamanGraphs.wl"]
Get["Calligraphs.wl"]
```

In[6]:=
```
(*
 * Calligraphs.wl was tested with Mathematica 12.1.
 * For some older versions a workaround might be needed (see section on input data).
 *)
```

Basics

In[7]:=
```
(*
 * The main purpose of the package is to compute the number of realizations
 * of a minimally rigid graph.
 * A graph in this package can always be given as a set of edges.
 * Edges can either be a given by two element lists {v1,v2}
 * or by Mathematica edges v1⟷v2
 *)
graph={{1,4},{1,5},{1,6},{2,3},{2,5},{2,6},{3,4},{3,6},{4,5}};
RealizationCountCS[graph]
```

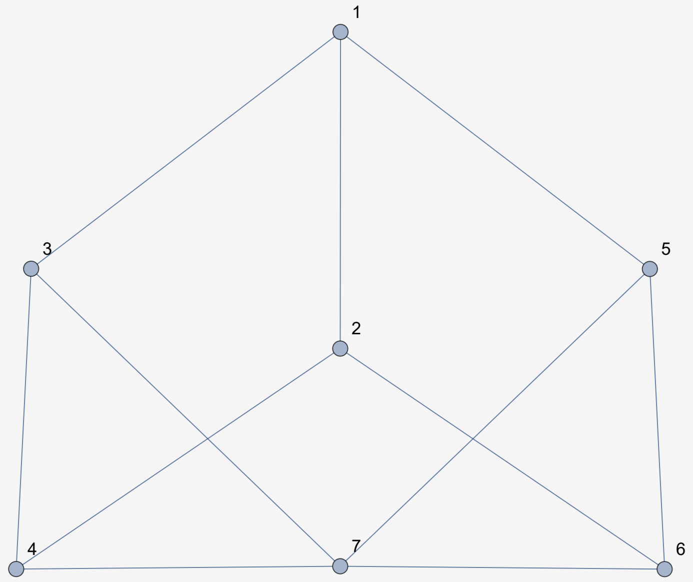
Out[8]= 24

In[9]:=
```
(* Lists of minimally rigid graphs can be found in [3] *)
```

In[10]:=

```
(*
 * For small graphs (like above) a previous algorithm is used (see [1,2]).
 * The procedure of splitting a mnimally rigid graph into calligraphs starts
 * whenever the number of vertices is high enough.
 *)
graph={{1,2},{1,3},{1,5},{2,4},{2,6},{3,4},{3,7},{4,7},{5,6},{5,7},{6,7}};
Graph[graph,VertexLabels->"Name"]
MinimallyRigidGraphQ[graph]
RealizationCountCS[graph]
```

Out[11]=



Out[12]= True

Out[13]= 56

In[14]:=

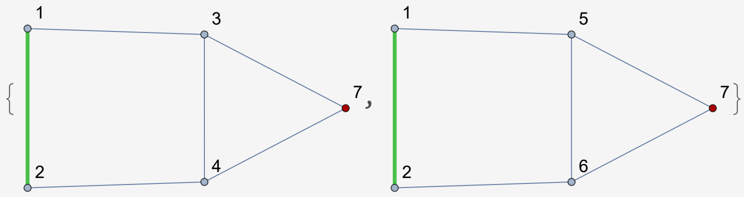
```
(*
 * For a minimally rigid graph we can check whether it has a non-trivial
 * calligraphic split.
 * Note that, internally this check actually determines the extistence of
 * a calligraphic split by finding one.
 *)
SplittableGraphQ[graph]
```

Out[14]= True

In[15]:=
```
(*
 * The output of splitting algorithms are in terms of {E,e,v},
 * where E is a set of edges, e is the edge representing the anchor
 * and v is the moving vertex.
 * Calligraphic splits can be visualized easily.
 *)
{csplit1,csplit2}=FindCalligraphicSplit[graph]
VisualizeSplit[%]
```

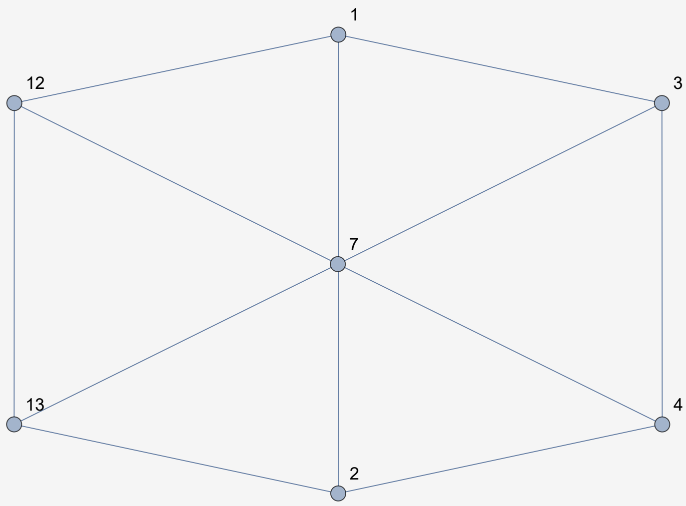
Out[15]=
```
{{{{1, 3}, {2, 4}, {3, 4}, {3, 7}, {4, 7}, {1, 2}}, {1, 2}, 7},
 {{{1, 2}, {1, 5}, {2, 6}, {5, 6}, {5, 7}, {6, 7}}, {1, 2}, 7}}
```

Out[16]=



In[17]:=
```
(*
 * The more or less inverse operation of a calligraphic split
 * is the union of two calligraphs.
 * Note that the vertices do not need to be different but the moving vertex
 * and the anchor need to be specified in the calligraph as before.
 *)
CalligraphUnion[csplit1,csplit2]
```

Out[17]=



In[18]:=
```
(*
 * The class of a calligraph {G,e,v},
 * where G is a minimally rigid graph minus one edge, e is an edge of the graph
 * and v a vertex of G that is not part of e.
 * Note that this is a more general description of the calligraph than in the paper.
 *)
csplit1
cclass1=CalligraphClass[csplit1]
csplit2
cclass2=CalligraphClass[csplit2]
```

Out[18]= {{{1, 3}, {2, 4}, {3, 4}, {3, 7}, {4, 7}, {1, 2}}, {1, 2}, 7}

Out[19]= {6, 2, 2}

Out[20]= {{{1, 2}, {1, 5}, {2, 6}, {5, 6}, {5, 7}, {6, 7}}, {1, 2}, 7}

Out[21]= {6, 2, 2}

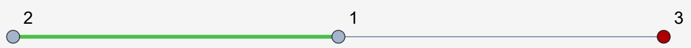
In[22]:=
```
(*
 * The product of the two classes yields the realization count.
 *)
ClassProduct[cclass1,cclass2]
RealizationCountCS[graph]
```

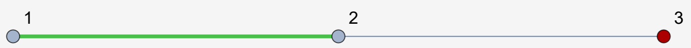
Out[22]= 56

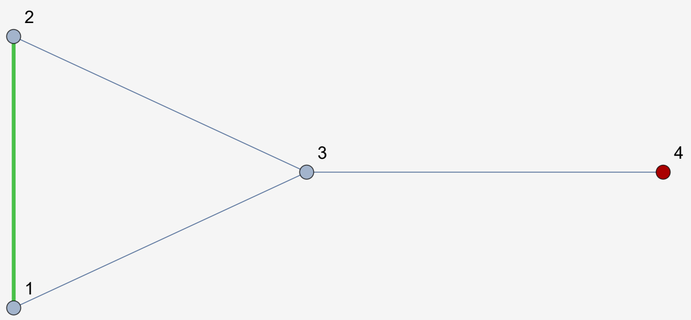
Out[23]= 56

Simple Calligraphs

In[24]:=
```
(*
 * Simple calligraphs
*)
cgraphL={{{1,2},{1,3}},{1,2},3};
VisualizeCalligraph[cgraphL]
cgraphR={{{1,2},{2,3}},{1,2},3};
VisualizeCalligraph[cgraphR]
cgraphC={{{1,2},{1,3},{2,3},{3,4}},{1,2},4};
VisualizeCalligraph[cgraphC]
```

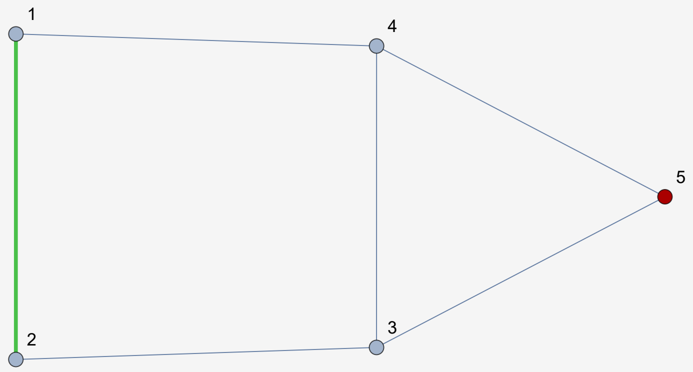
Out[25]=


Out[27]=


Out[29]=


In[30]:=
```
(*
 * We can check whether a given graph together with an edge
 * and a vertex is indeed a calligraph.
 * Calligraphs can be visualized.
 * Note that the result is not a calligraph as a data format.
 *)
cg={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{1,2},5};
VisualizeCalligraph[cg]
CalligraphQ[cg]
cg2={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{2,1},5};
CalligraphQ[cg2]
CalligraphQ[cgraphL]
CalligraphQ[cgraphR]
CalligraphQ[cgraphC]
```

Out[31]=



Out[32]= True

Out[34]= True

Out[35]= True

Out[36]= True

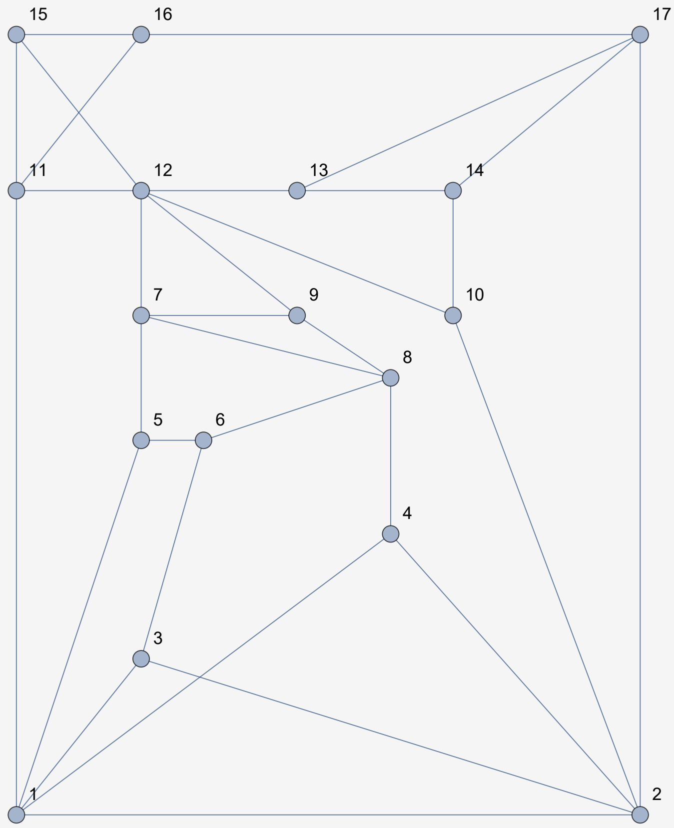
Out[37]= True

Example from Paper

In[38]:=
```
(*
 * This example cannot be computed in reasonable time with the fallback algorithm
 * but it can be computed with the algorithm from this package.
 *)
pex={{1,2},{1,3},{1,4},{1,5},{1,11},{2,3},{2,4},{2,10},{2,17},{3,6},{4,8},
     {5,6},{5,7},{6,8},{7,8},{7,9},{7,12},{8,9},{9,12},{10,12},{10,14},{11,12},
     {11,15},{11,16},{12,13},{12,15},{13,14},{13,17},{14,17},{15,16},{16,17}}
Graph[pex,
  VertexLabels->"Name",
  VertexCoordinates->{{0,0},{2,0},{0.4,0.5},{1.2,0.9},{0.4,1.2},{0.6,1.2},
                      {0.4,1.6},{1.2,1.4},{0.9,1.6},{1.4,1.6},{0,2},{0.4,2},
                      {0.9,2},{1.4,2},{0,2.5},{0.4,2.5},{2,2.5}}
]
RealizationCountCS[pex]
```

Out[38]=
```
{{1, 2}, {1, 3}, {1, 4}, {1, 5}, {1, 11}, {2, 3}, {2, 4}, {2, 10},
 {2, 17}, {3, 6}, {4, 8}, {5, 6}, {5, 7}, {6, 8}, {7, 8}, {7, 9}, {7, 12},
 {8, 9}, {9, 12}, {10, 12}, {10, 14}, {11, 12}, {11, 15}, {11, 16},
 {12, 13}, {12, 15}, {13, 14}, {13, 17}, {14, 17}, {15, 16}, {16, 17}}
```

Out[39]=



Out[40]=
```
200192
```

Input Data

In[41]:=

```
(*
 * Most functions of the package can deal with graphs being represented by
 * lists of edges, where edges are given as two-element lists.
 * However, the functions also take Mathematica's Graph and UndirectedEdge data types.
 * This might be needed for some older versions of Mathematica.
 *)
```

In[42]:=

```
cg={{{1,2},{1,3}},{1,2},3}
CalligraphClass[cg]
gcg=ListToCGraph[cg]
CalligraphClass[gcg]
```

Out[42]= {{{1, 2}, {1, 3}}, {1, 2}, 3}

Out[43]= {1, 1, 0}

Out[44]= {{1 ⟷ 2, 1 ⟷ 3}, 1 ⟷ 2, 3}

Out[45]= {1, 1, 0}

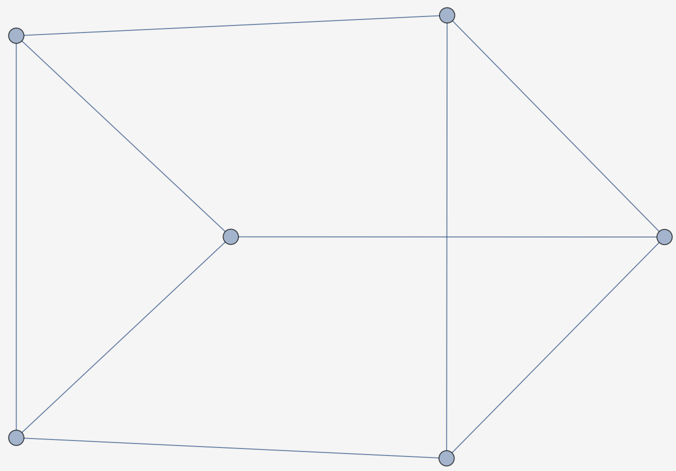
In[46]:=

```
graph={{1,4},{1,5},{1,6},{2,3},{2,5},{2,6},{3,4},{3,6},{4,5}}
RealizationCountCS[graph]
ggraph=Graph[graph]
RealizationCountCS[ggraph]
```

Out[46]= {{1, 4}, {1, 5}, {1, 6}, {2, 3}, {2, 5}, {2, 6}, {3, 4}, {3, 6}, {4, 5}}

Out[47]= 24

Out[48]=



Out[49]= 24

Advanced

In[50]:=
```
(*
 * We can ask the algorithm to use splits
 * for which the two calligraphs have a vertex count that is as close as possible,
 * with the disadvantage that we first need to find all splits
 * and then choose the best one.
 * We see that for the current example this causes too much overhead
 * (i.e. more time is needed for the search than saved by using a possibly better spli
 *)
RealizationCountCS[pex]//Timing
RealizationCountCS[pex,SplittingAlgorithm->FindBalancedCalligraphicSplit]//Timing
```

Out[50]= {5.95313, 200 192}

Out[51]= {7.625, 200 192}

In[52]:=
```
(*
 * We can use the balanced splitting only for larger graphs, if we want.
 *)
RealizationCountCS[pex,
  SplittingAlgorithm->FindBalancedThresholdCalligraphicSplit,
  BalanceThreshold->6
]//Timing
RealizationCountCS[pex,
  SplittingAlgorithm->FindBalancedThresholdCalligraphicSplit,
  BalanceThreshold->16
]//Timing
```

Out[52]= {6.0625, 200 192}

Out[53]= {6.23438, 200 192}

In[54]:=
```
(*
 * The fallback algorithm is used when the input graph is not splittable and
 * when there is a small number of vertices.
 * We can decide the threshold on the vertex count
 * for which the fallback algorithm is used.
 * The default is 6.
 *)
RealizationCountCS[pex]//Timing
RealizationCountCS[pex,BaseDeg->8]//Timing
```

Out[54]= {5.79688, 200 192}

Out[55]= {6.375, 200 192}

In[56]:=

```
(*
 * We can turn on error messages to see why an input is not a calligraph.
 *)
ncg={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{1,3},5}
CalligraphQ[ncg,ShowMessages->True]
ncg={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{1,2},0};
CalligraphQ[ncg,ShowMessages->True]
ncg={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{3,5},5};
CalligraphQ[ncg,ShowMessages->True]
```

Out[56]= `{{{1, 2}, {1, 4}, {2, 3}, {3, 4}, {3, 5}, {4, 5}}, {1, 3}, 5}`

CalligraphQ::notanedge: Position 2 in {{{1, 2}, {1, 4}, {2, 3}, {3, 4}, {3, 5}, {4, 5}}, {1, 3}, 5} is not an edge in the graph.

Out[57]= `False`

CalligraphQ::notavertex: Position 3 in {{{1, 2}, {1, 4}, {2, 3}, {3, 4}, {3, 5}, {4, 5}}, {1, 2}, 0} is not a vertex of the graph.

Out[59]= `False`

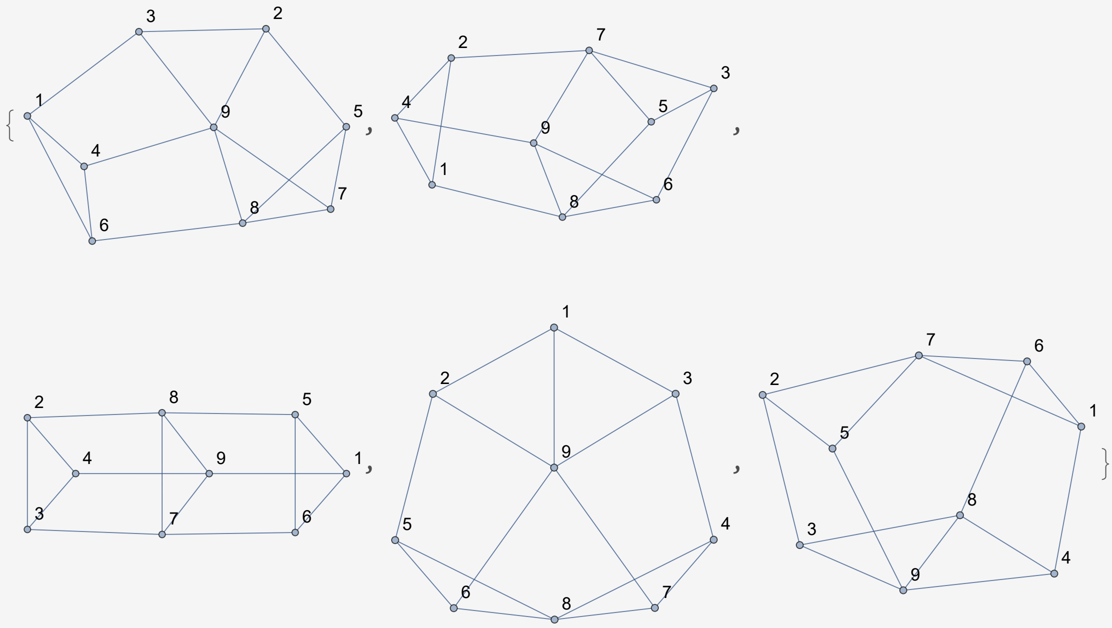
CalligraphQ::notacgraph: The input is not a calligraph since the vertex lies in a common minimally rigid subgraph with the edge.

Out[61]= `False`

In[62]:=

```
(*
 * For getting constructed examples of splittable graphs
 * we can glue two minimally rigid graphs.
 * Different gluing results in possibly different
 * number of realizations of the new graph.
 * By default it is gluing on the first edge from each of the graphs and
 * takes all possible vertices as a moving vertex.
 *)
graph={{1,4},{1,5},{1,6},{2,3},{2,5},{2,6},{3,4},{3,6},{4,5}};
GlueGraphs[graph,graph,VertexLabels->"Name"]
RealizationCountCS/@%
```

Out[63]=



Out[64]= `{192, 288, 288, 256, 288}`

In[65]:=
```
(* If the input are not calligraphs we get an error in CalligraphUnion. *)
cg={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{1,2},5};
ncg={{{1,2},{1,4},{2,3},{3,4},{3,5},{4,5}},{3,5},5};
CalligraphUnion[cg,ncg]
```

··· CalligraphUnion::notacg: Input {{{1, 2}, {1, 4}, {2, 3}, {3, 4}, {3, 5}, {4, 5}}, {3, 5}, 5} is not a calligraph.

References

In[68]:=

```
(* [1] J. Capco, M. Gallet, G. Grasegger, C. Koutschan, N. Lubbes, and J. Schicho.
 *     An algorithm for computing the number of realizations of a Laman graph, 2018
 *     doi: 10.5281/zenodo.1245506
 *     Implementing [2]
 * [2] J. Capco, M. Gallet, G. Grasegger, C. Koutschan, N. Lubbes, and J. Schicho.
 *     The number of realizations of a Laman graph.
 *     SIAM Journal on Applied Algebra and Geometry, 2(1):94-125, 2018
 *     doi: 10.1137/17M1118312
 * [3] J. Capco, M. Gallet, G. Grasegger, C. Koutschan, N. Lubbes, and J. Schicho.
 *     The number of realizations of all Laman graphs with at most 12 vertices
 *     Dataset, doi:10.5281/zenodo.1245517
 *)
```