

# **Pajek**

**Program for Analysis and  
Visualization of Large Networks**

**Reference Manual**

List of commands with short explanation  
version 1.22

**Vladimir Batagelj and Andrej Mrvar**

**Ljubljana, January 16, 2008**

©1996, 2008 V. Batagelj, A. Mrvar. Free for noncommercial use.  
PdfLaTeX version October 1, 2003

**Vladimir Batagelj**  
**Department of Mathematics, FMF**  
**University of Ljubljana, Slovenia**

<http://vlado.fmf.uni-lj.si/>

[vladimir.batagelj@fmf.uni-lj.si](mailto:vladimir.batagelj@fmf.uni-lj.si)

**Andrej Mrvar**  
**Faculty of Social Sciences**  
**University of Ljubljana, Slovenia**

<http://mrvar.fdv.uni-lj.si/>

[andrej.mrvar@fdv.uni-lj.si](mailto:andrej.mrvar@fdv.uni-lj.si)

# Contents

<b>1</b>	<b>Pajek</b>	<b>3</b>
<b>2</b>	<b>Data objects</b>	<b>6</b>
<b>3</b>	<b>Main Window Tools</b>	<b>8</b>
3.1	File . . . . .	8
3.2	Net . . . . .	12
3.3	Nets . . . . .	29
3.4	Operations . . . . .	32
3.5	Partition . . . . .	41
3.6	Partitions . . . . .	42
3.7	Vector . . . . .	43
3.8	Vectors . . . . .	44
3.9	Permutation . . . . .	45
3.10	Permutations . . . . .	46
3.11	Cluster . . . . .	46
3.12	Hierarchy . . . . .	46
3.13	Options . . . . .	46
3.14	Info . . . . .	49
3.15	Tools . . . . .	50
<b>4</b>	<b>Draw Window Tools</b>	<b>53</b>
4.1	Main Window Draw Tool . . . . .	53
4.2	Layout . . . . .	53
4.3	Layers . . . . .	55
4.4	GraphOnly . . . . .	57
4.5	Previous . . . . .	57
4.6	Redraw . . . . .	57
4.7	Next . . . . .	57
4.8	Options . . . . .	57
4.9	Export . . . . .	61
4.10	Spin . . . . .	64
4.11	Move . . . . .	64
4.12	Info . . . . .	64
<b>5</b>	<b>Exports to EPS/SVG/X3D/VRML</b>	<b>65</b>
5.1	Defaults . . . . .	65
5.2	Parameters in EPS, SVG, X3D, and VRML Defaults Window . . .	65
5.3	Exporting pictures to EPS/SVG – defining parameters in input file	69

<b>6</b>	<b>Using Macros in Pa jek</b>	<b>74</b>
6.1	What is a Macro? . . . . .	74
6.2	How to record a Macro? . . . . .	74
6.3	How to execute the Macro? . . . . .	74
6.4	Example . . . . .	74
6.5	List of macros available in installation file . . . . .	75
6.5.1	Macros prepared for genealogies and other acyclic networks	75
6.5.2	Macros prepared for computing derived kinship relations .	76
6.6	Repeating last command . . . . .	76
<b>7</b>	<b>Blockmodeling in Pa jek</b>	<b>78</b>
7.1	MDL files . . . . .	78
7.2	Examples of MDL files . . . . .	80
7.2.1	Regular blocks . . . . .	80
7.2.2	Diagonal blocks (clustering) . . . . .	80
7.2.3	Acyclic model (up) . . . . .	80
7.2.4	Acyclic model with symmetric clusters (down) . . . . .	80
7.2.5	Center-Periphery . . . . .	81
7.2.6	Regular path . . . . .	81
7.2.7	Regular chain . . . . .	81
7.2.8	2-mode 'standard model' for Davis.net . . . . .	82
<b>8</b>	<b>Colors in Pa jek</b>	<b>83</b>
<b>9</b>	<b>Citing Pa jek</b>	<b>85</b>

# 1 Pajek



**Pajek** is a program, for Windows, for analysis and visualization of *large networks* having some thousands or even millions of vertices. In Slovenian language the word *pajek* means spider. The latest version of **Pajek** is freely available, for noncommercial use, at its home page:

<http://vlado.fmf.uni-lj.si/pub/networks/pajek/>

We started the development of **Pajek** in November 1996. **Pajek** is implemented in Delphi (Pascal). Some procedures were contributed by Matjaž Zaveršnik.

The main motivation for development of **Pajek** was the observation that there exist several sources of large networks that are already in machine-readable form. **Pajek** should provide tools for analysis and visualization of such networks: collaboration networks, organic molecule in chemistry, protein-receptor interaction networks, genealogies, Internet networks, citation networks, diffusion (AIDS, news, innovations) networks, data-mining (2-mode networks), etc. See also collection of large networks at:

<http://vlado.fmf.uni-lj.si/pub/networks/data/>

The design of **Pajek** is based on our previous experiences gained in development of graph data structure and algorithms libraries Graph and X-graph, collection of network analysis and visualization programs STRAN, RelCalc, Draw, Energ, and SGML-based graph description markup language NetML.

<http://vlado.fmf.uni-lj.si/pub/networks/default.htm>

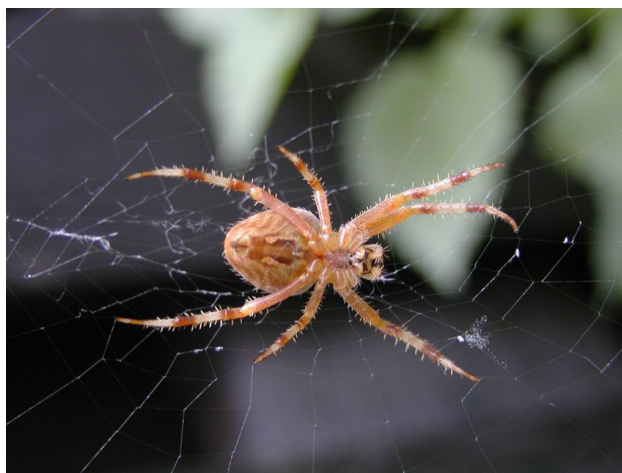


Figure 1: *Pajek/Spider*

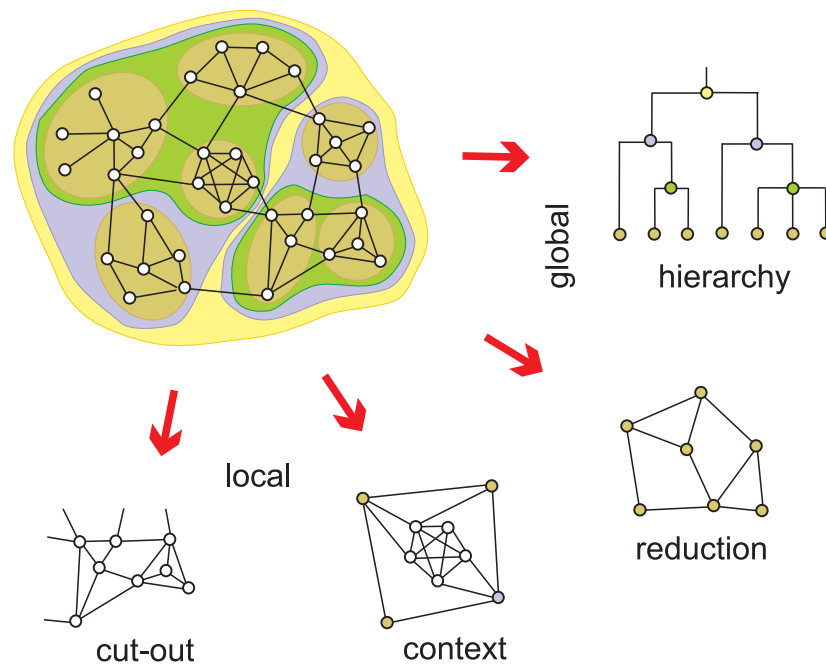


Figure 2: Approaches to deal with large networks

The main goals in the design of **Pajek** are:

- to support abstraction by (*recursive*) *decomposition* of a large network into several smaller networks that can be treated further using more sophisticated methods;
- to provide the user with some powerful *visualization* tools;
- to implement a selection of *efficient (subquadratic) algorithms* for analysis of large networks.

With **Pajek** we can: *find* clusters (components, neighbourhoods of ‘important’ vertices, cores, etc.) in a network, *extract* vertices that belong to the same clusters and *show* them separately, possibly with the parts of the context (detailed local view), *shrink* vertices in clusters and show relations among clusters (global view).

Besides ordinary (directed, undirected, mixed) networks **Pajek** supports also *multi-relational networks*, *2-mode networks* (bipartite (valued) graphs – networks between two disjoint sets of vertices), and *temporal networks* (dynamic graphs – networks changing over time).

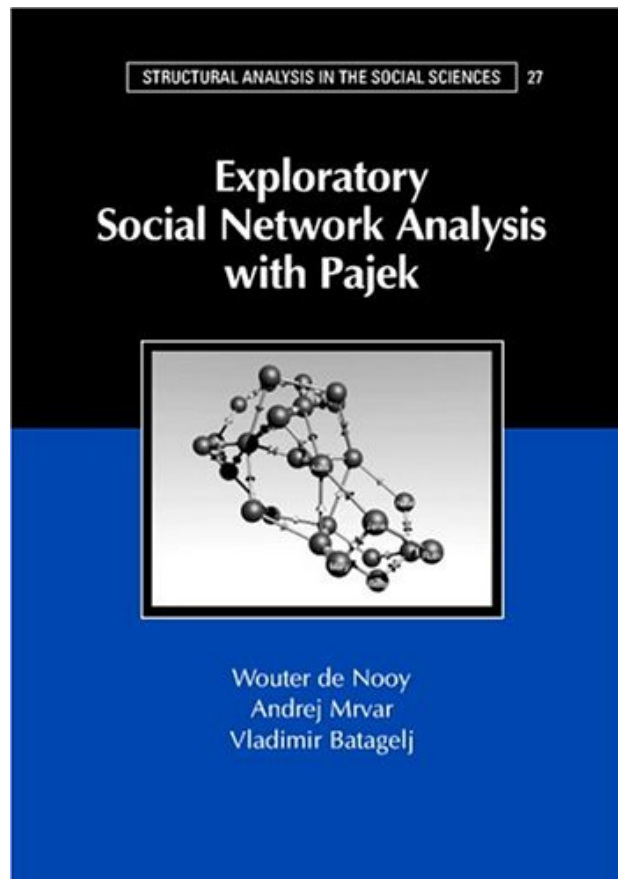


Figure 3: *Pajek* textbook

This manual provides short explanations of all procedures implemented in the last version of **Pajek**. The novice users we advise to read the **Pajek** textbook [25]

de Nooy W., Mrvar A., Batagelj V. (2002) *Exploratory Social Network Analysis With Pajek*. Structural Analysis in the Social Sciences 27, Cambridge University Press, 2005.

For an overview of *network analysis with Pajek* see the NICTA workshop slides [4].

## 2 Data objects

In **Pajek** six types of objects are used:

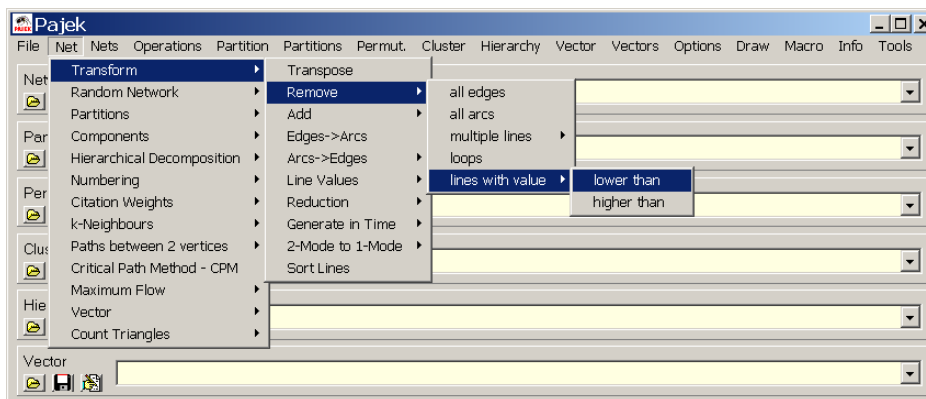


Figure 4: **Pajek**'s Main Window

1. **Networks** – main objects (vertices and lines). Default extension: **.net**. Network can be presented on input file in different ways:

- using arcs/edges (e.g. 1 2 – line from 1 to 2)
- using arclists/edgeslists (e.g. 1 2 3 – line from 1 to 2 and from 1 to 3)
- matrix format
- UCINET, GEDCOM, chemical formats. . .

Additional information for network drawing can be included in input file as well. This is explained in the section [Exports to EPS/SVG/VRML](#).

2. **Partitions** – they tell for each vertex to which class vertex belong. Default extension: **.clu**.
3. **Permutations** – reordering of vertices. Default extension: **.per**.
4. **Clusters** – subset of vertices (e.g. one class from partition). Default extension: **.cls**.
5. **Hierarchies** – hierarchically ordered vertices. Example:

```

Root
  g1      g2
  g11 g12  v5, v6, v7
  v1, v2  v3, v4

```



Root has two subgroups – g1 and g2. g2 is a leaf – cluster with vertices v5,v6 and v7. g1 has two subgroups – g11 and g12...Default extension: `.hie`.

6. **Vectors** – they tell for each vertex some numerical property (real number). Default extension: `.vec`.

By double clicking on selected network, partition,... you can show the object on screen.

The procedures in **Pajek**'s main window (see Figure 4) are organized according to the types of data objects they use as input.

Permutations, partitions and vectors can be used to store properties of vertices measured in different scales: ordered, nominal (categorical) and numeric.



Figure 5: *Spider web*; Photo: Vladimir Batagelj.

## 3 Main Window Tools

### 3.1 File

Input/Output manipulation with the six data objects.

- **Network – N**
  - **Read** network from Ascii file.
  - **Edit** network. Choose vertex, show its neighbors and then:
    - \* add new lines to/from selected vertex (by left mouse double clicking on Newline);
    - \* delete lines (by left mouse double clicking);
    - \* change value of line (by single right mouse clicking);
    - \* subdivide line to two orthogonal lines using new invisible vertex (by single middle mouse clicking).
  - **Save** selected network to Ascii file.
  - **Export Matrix to EPS** – write matrix in EPS format:
    - \* **Original** – using default numbering (for 1-mode and 2-mode networks).
    - \* **Using Permutation** – using current permutation. Additionally lines can be drawn to divide different classes defined by selected partition. Option can be used for 1-mode and 2-mode networks.
    - \* **Using Partition** – using current partition. In the text window number and density of lines among classes (and vertices in selected two classes) are displayed. Additionally matrix is exported to EPS where density is expressed using shadowing:
      1. **Structural** – Densities are normalized according to maximum possible number of lines among classes (suitable for dense networks).
      2. **Delta** – Densities are normalized according to vertices having the highest number of *input* and *output* neighbors in classes (suitable for sparse networks).
    - \* **Diamonds for Negative Values** – If checked squares are used for positive values, diamonds for negative (useful for black and white printing).
    - \* **Only Black Borders** – If checked all squares in matrix will have black borders, otherwise dark squares will have white and light squares will have black borders.

Table 1: List of time events.

Event	Explanation
TI $t$	initial events – following events happen when time point $t$ starts
TE $t$	end events – following events happen when time point $t$ is finished
AV $vns$	add vertex $v$ with label $n$ and properties $s$
HV $v$	hide vertex $v$
SV $v$	show vertex $v$
DV $v$	delete vertex $v$
AA $uvs$	add arc $(u,v)$ with properties $s$
HA $uv$	hide arc $(u,v)$
SA $uv$	show arc $(u,v)$
DA $uv$	delete arc $(u,v)$
AE $uvs$	add edge $(u:v)$ with properties $s$
HE $uv$	hide edge $(u:v)$
SE $uv$	show edge $(u:v)$
DE $uv$	delete edge $(u:v)$
CV $vs$	change vertex property – change property of vertex $v$ to $s$
CA $uvs$	change arc property – change property of arc $(u,v)$ to $s$
CE $uvs$	change edge property – change property of edge $(u:v)$ to $s$
CT $uv$	change type – change (un)directedness of line $(u,v)$
CD $uv$	change direction of arc $(u,v)$
PE $uvs$	replace pair of arcs $(u,v)$ and $(v,u)$ by single edge $(u:v)$ with properties $s$
AP $uvs$	add pair of arcs $(u,v)$ and $(v,u)$ with properties $s$
DP $uv$	delete pair of arcs $(u,v)$ and $(v,u)$
EP $uvs$	replace edge $(u:v)$ by pair of arcs $(u,v)$ and $(v,u)$ with properties $s$

- **Change Label** of selected network.
- **Dispose** selected network from memory.

- **Time Events Network – N**

- **Read Time Events** – Read time network described using time events. See Table 1.

List of properties  $s$  can be empty as well. If several edges (arcs) can connect two vertices, additional tag like  $:k$  ( $k$ -th line) must be given to determine to which line the command applies. E.g. command HE : 3 14 37 results in hiding the third edge connecting vertices 14 and 37. Example of time network described using time events:

```

*Vertices 3
*Events
TI 1
AV 2 "b"
TE 3
HV 2
TI 4
AV 3 "e"
TI 5
AV 1 "a"
TI 6
AE 1 3 1
TI 7
SV 2
AE 1 2 1
TE 7
DE 1 2
DV 2
TE 8
DE 1 3
TE 10
HV 1
TI 12
SV 1
TE 14
DV 1

```

See also other possibility: description of time network using **time intervals**.

– **Save** – Save time network in time events format.

- **Partition – C**

- **Read** partition from Ascii file.
- **Edit** partition (put vertices to classes).
- **Save** selected partition to Ascii file.
- **Change Label** of selected partition.
- **Dispose** selected partition from memory.

- **Permutation – P**

- **Read** permutation from Ascii file.
- **Edit** permutation (interchange positions of two vertices).
- **Save** selected permutation to Ascii file.
- **Change Label** of selected permutation.
- **Dispose** selected permutation from memory.

- **Cluster – S** (list of selected vertices)

- **Read** cluster from Ascii file.
- **Edit** cluster (add and delete vertices).
- **Save** selected cluster to Ascii file.
- **Change Label** of selected cluster.
- **Dispose** selected cluster from memory.
- **Hierarchy – H**
  - **Read** hierarchy from Ascii file.
  - **Edit** hierarchy (change types and names of nodes, or show vertices (and subtree) belonging to selected node). Nodes can be pushed up and down within hierarchy.
  - **Save** selected hierarchy to Ascii file.
  - **Change Label** of selected hierarchy.
  - **Dispose** selected hierarchy from memory.
- **Vector – V**
  - **Read** vector from Ascii file.
  - **Edit** vector (change components of vector).
  - **Save** selected vector(s) to Ascii file. If cluster representing vector id's is present, all vectors with corresponding id numbers will be saved to the same output file. Vector's id can be added to cluster by pressing *V* on the selected vector (empty cluster should be created first). All vectors must have the same dimensions.
  - **Change Label** of selected vector.
  - **Dispose** selected vector from memory.
- **Pajek Project File – \*.paj**
  - **Read Pajek** project file (file containing all possible **Pajek** data objects – networks, partitions, permutations, clusters, hierarchies and vectors).
  - **Save** all currently loaded objects as a **Pajek** project file.
- **Repeat session** – During program execution all commands are written to file `*.log`. In this way you can repeat any execution by running selected log file. If you change in the log file a name of a file to `?`, program will ask for name when running logfile next time (so you can repeat the same

sequence of steps – logfile with different input data). If startup logfile (Pajek.log) exists (in the same directory as Pajek.exe), it is automatically executed every time when **Pajek** is run.

- **Show Report Window** – Bring the report window in the front in the case that it was closed or is not visible.
- **Exit** program.

## 3.2 Net

Operations, for which only a network is needed as input.

- **Transform**
  - **Transpose** – Transposed network of selected network:
    - \* **1-Mode** - Change direction of arrows.
    - \* **2-Mode** - Interchange Rows and Cols.
  - **Remove**
    - \* **Selected Vertices** – Remove selected vertices from network.
    - \* **all Edges** – Remove all edges from selected network.
    - \* **all Arcs** – Remove all arcs from selected network.
    - \* **Multiple Lines** – Remove all multiple lines from selected network.
      1. **Sum Values** – Values of all deleted lines are added to not deleted line between corresponding two vertices.
      2. **Number of Lines** – Value of line between two vertices in a new network correspond to the number of lines between the two vertices in original network.
      3. **Min Value** – Minimum value of all lines between two vertices is selected.
      4. **Max Value** – Maximum value of all lines between two vertices is selected.
      5. **Single Line** – Value of line between two vertices in a new network is 1.
    - \* **Loops** – Remove all loops from selected network.
    - \* **Lines with Value**
      1. **lower than** – Remove all lines with value lower than specified value.

2. **higher than** – Remove all lines with value higher than specified value.
  3. **within interval** – Remove all lines with values within specified interval.
- \* **all Arcs from each Vertex except**
    1. **k with Lowest Line Values** – Sort lines around vertices in ascending order according to output line values. Keep only selected number of lines with lowest values.
    2. **k with Highest Line Values** – Sort lines around vertices in descending order according to output line values. Keep only selected number of lines with highest values.
- **Add** additional vertices, lines or vertices/lines labels to network.
- \* **Vertices** – Copy network to new network. Dimension can be enlarged for selected number of vertices (additional vertices without lines are added).
  - \* **Source and Sink** – If network is acyclic, add unique first and last vertex (new network has two artificial vertices).
  - \* **Default Vertex Labels** – Replace current labels of vertices with default vertices labels (v1, v2...).
  - \* **Vertex Labels from File** – Change the default vertices labels (v1, v2...) with labels given in input network file.
  - \* **Line Labels as Line Values** – replace labels of lines (or create new if there are no) with line values. Number of decimal places is the same as used in Draw window for marking lines with line values.
  - \* **Sibling edges** – Add sibling edges to vertices with a common
    1. **Input** – arc-ancestor
    2. **Output** – arc-descendant
- **Edges** → **Arcs** – Convert all edges to arcs (in both directions) (make directed network).
- **Arcs** → **Edges**
- \* **All** – Convert all arcs to edges (make undirected network).
  - \* **Bidirected only** – Convert only arcs in both directions to edges:
    1. **Sum Values** – Value of the new edge is the sum of values of both arcs.
    2. **Min Value** – Value of the new edge is the smaller of values of arcs.

3. **Max Value** – Value of the new edge is the larger of values of arcs.

– **Bidirected Arcs** → **Arc**

- \* **Select Min Value** – If there exist bidirected arcs between two vertices retain only the arc with lower value and remove the arc with higher value. If both values are equal replace both arcs with an edge.
- \* **Select Max Value** – If there exist bidirected arcs between two vertices retain only the arc with higher value and remove the arc with lower value. If both values are equal replace both arcs with an edge.

– **Line Values** – Transformations of line values:

- \* **Recode** – Display frequency distribution of line values according to selected intervals and recode line values in this way.
- \* **Multiply by** a constant.
- \* **Add Constant** to line values.
- \* **Absolute** line values.
- \* **Absolute + Sqrt** – square root of line values.
- \* **Truncate** – truncated line values.
- \* **Exp** – exponent of line values.
- \* **Ln** – natural logarithm of line values.
- \* **Power** – selected power of line values.
- \* **Normalize**
  1. **Sum** – normalize so that the sum of line values will be 1
  2. **Max** – normalize so that the maximum line value will be 1

– **Reduction**

- \* **Degree** – (Recursively) delete from network all vertices with degree lower than selected value (according to Input, Output or All degree). Operation can be limited to selected cluster.
- \* **Hierarchical** – Recursively delete from network all vertices that have only 0 or 1 neighbor. Results: simpler network and hierarchy with deleted vertices. Original network can be later restored (if we forget directions of lines).
- \* **Subdivisions** – Recursively delete from network all vertices that have exactly 2 neighbors (together with corresponding two lines) and (instead of that) add direct line between these two neighbors.



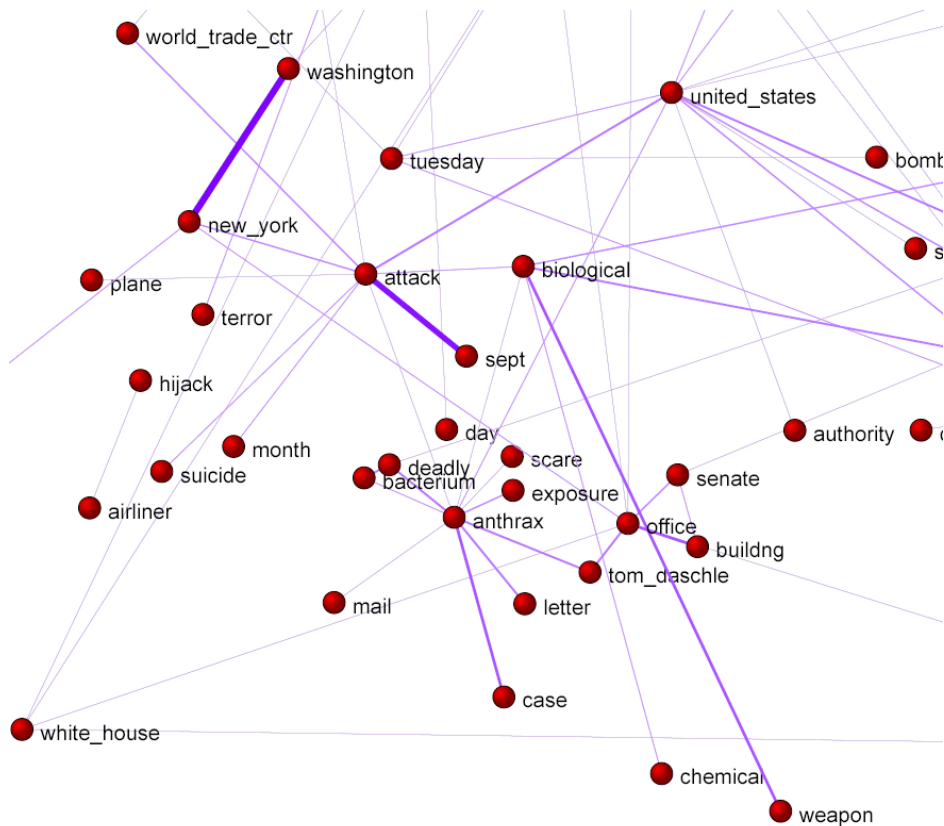


Figure 6: Part of Reuters Terror News network on the 36th day.

Result is simpler network (for drawing). Original network cannot be restored!

\* **Design (flow graph)** Reduction of all structural parts of network according to McCabe (for programs – flow graphs) [42].

– **Generate in Time** – Generate network in specified time(s) or interval. Input first time, last time and step (integers).

Additional parameters when vertices and lines are active should be given in network to perform this operation. They must be given between signs [ and ]:

– is used to divide lower and upper limit of interval,

, is used to separate intervals,

\* means infinity. Example:

```
*Vertices 3
1 "a" [5-10,12-14]
2 "b" [1-3,7]
```

```

3 "e" [4-*]
*Edges
1 2 1 [7]
1 3 1 [6-8]

```

Vertex 'a' is active from times 5 to 10, and 12 to 14, vertex 'b' in times 1 to 3 and in time 7, vertex 'e' from time 4 on. Line from 1 to 2 is active only in time 7, line from 1 to 3 in times 6 to 8.

The lines and vertices in a temporal network should satisfy the *consistency* condition: if a line is active in time  $t$  then also its end-vertices are active in time  $t$ . When generating time slices of a given temporal network only 'consistent' lines are generated.

Note that time records should always be written as last in the row where vertices / lines are defined.

See also other possibility of describing time network: description of time network using **time events**.

- \* **All** – Generate all networks in specified times.
  - \* **Only Different** – Generate network in specified time only if the new network will differ in at least one vertex or line from the last network which was generated.
  - \* **Interval** – Generate network with vertices and lines present in selected interval.
- **1-Mode to 2-Mode** – Generate 2-mode network from any network.
  - **2-Mode to 1-Mode** – Generate an ordinary (1-mode) network from 2-mode (affiliation) network. Result is a valued network. To store a 2-mode network in input file use **Pajek** or Ucinet format (look at **Davis.dat** from Ucinet dataset).
    - \* **Rows** – Result is a network with relations among row elements (actors). The value of line tells number of common events of the two actors.
    - \* **Columns** – Result is network with relations among column elements (events). The value of a line tells number of actors that took part in both events.
    - \* **Include Loops** – If checked, loops with value telling the total number of events for each actor (total number of actors for each event), are added.
    - \* **Multiple Lines** – Generate nonvalued 1-mode network, where multiple lines among vertices can exist. The label of the generated line corresponds to the label of the event/actor that served

to induce the line. If partition of the same dimension is present, multirelational network can be generated.

- \* **Normalize 1-Mode** – Normalize the obtained 1-Mode network. 1-Mode network must be obtained with option **include loops** checked, and **multiple lines** not checked:

$$\text{Geo}_{ij} = \frac{a_{ij}}{\sqrt{a_{ii}a_{jj}}}$$

$$\text{Input}_{ij} = \frac{a_{ij}}{a_{jj}}$$

$$\text{Output}_{ij} = \frac{a_{ij}}{a_{ii}}$$

$$\text{Min}_{ij} = \frac{a_{ij}}{\min(a_{ii}, a_{jj})}$$

$$\text{Max}_{ij} = \frac{a_{ij}}{\max(a_{ii}, a_{jj})}$$

$$\text{MinDir}_{ij} = \begin{cases} \frac{a_{ij}}{a_{ii}} & a_{ii} \leq a_{jj} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{MaxDir}_{ij} = \begin{cases} \frac{a_{ij}}{a_{jj}} & a_{ii} \leq a_{jj} \\ 0 & \text{otherwise} \end{cases}$$

The obtained network is usually not sparse. To make it sparser use **Net/Transform/Remove/lines with value/lower than**.

- \* **Rows=Cols** – Transform 2-Mode network with the same subsets of vertices to 1-Mode network.

#### – Multiple Relations

- \* **Extract Relation(s)** – Extract one or selected list of relations from selected multiple relations network.
- \* **Canonical Numbering** – Enumerate relations with sequential numbers 1, 2, ...
- \* **Generate 3-Mode Network** – generate a 3-mode network from 1-mode or 2-mode multirelational network. For each line in multirelational network  $r: i \ j \ v$  (line from  $i$  to  $j$  with value  $v$ , relation number is  $r$ ) generate the following three lines (triangle):
  - 1-mode networks:
    - $i \ N+j \ v$
    - $i \ 2N+r \ v$
    - $N+j \ 2N+r \ v$

· 2-mode networks:

```

i j v
i N+M+r v
j N+M+r v

```

where  $N$  is cardinality of the first mode and  $M$  cardinality of the second mode.

- \* **Line Values** – > **Relation Numbers** – Store line values as relation numbers (absolute truncated values).
  - \* **Relation Numbers** – > **Line Values** – Store relation numbers as line values.
  - \* **Change Relation Number - Label** – Change selected relation number to new relation number with corresponding label.
- **Sort Lines** –
- \* **Neighbors around Vertices** – For each vertex sort lines connected to it in ascending order according to other end-vertex.
  - \* **Line Values** – Sort lines in ascending or descending order according to line values.
- **Random Network** – Generate random network of selected dimension
    - **Total No. of Arcs** – Generate random directed network of selected dimension and given number of arcs.
    - **Vertices Output Degree** – Generate random directed network of selected dimension and output degree of each vertex in given range.
    - **Erdos-Renyi** – Generate undirected, directed, acyclic, bipartite or 2-mode random network according to model defined by Erdos and Renyi: each line is selected with the given probability  $p$ . Instead of  $p$ , which is for large and sparse networks (very) small number, in **Pa jek** a more intuitive *average degree*  $\bar{d}$  is used. They are connected with relations  $\bar{d} = \frac{1}{n} \sum_{v \in V} \deg(v) = \frac{2m}{n}$  and  $m = pM$  where  $n = |V|$ ,  $m = |L|$  and  $M$  is the number of lines in maximal possible network – for example, for undirected graphs  $M = n(n - 1)$ .
    - **Scale Free** – Generate scale free undirected, directed or acyclic network. The procedure is based on a refinement of the model for generating *scale free networks*, proposed in [47]. At each step of the growth a new vertex and  $k$  edges are added to the network  $N$ . The endpoints of the edges are randomly selected among all vertices according to the probability

$$\Pr(v) = \alpha \frac{\text{indeg}(v)}{|E|} + \beta \frac{\text{outdeg}(v)}{|E|} + \gamma \frac{1}{|V|}$$

where  $\alpha + \beta + \gamma = 1$ . It is easy to check that  $\sum_{v \in V} \Pr(v) = 1$ .

- **Extended Model** – Generate random network according to extended model defined by Albert and Barabasi [2].

- **Partitions** – Partitioning Network. Result is a Partition.

- **Degree**

- \* **Input** – Number of lines into vertices.
- \* **Output** – Number of lines out of vertices.
- \* **All** – Number of neighbors of vertices.

- **Domain** – For each vertex compute its domain according to *input*, *output* or *all* neighbors. Results are:

- \* Partition containing size of domain - number of reachable vertices.
- \* Vector containing the normalized size of domain - normalization is done by total number of vertices – 1.
- \* Vector containing the average distance from/to domain.

Proximity Prestige index can be computed by dividing the normalized size of domain by average distance.

- **Core** –  $k$ -core is a subnetwork of given network where each vertex has at least  $k$  neighbors in the same core according to:

- \* **Input** ... lines coming into vertex.
- \* **Output** ... lines going out of vertex.
- \* **All** ... all neighbors.
- \* **2-Mode** – core partition of a 2-mode network. Given minimum degree in first ( $k_1$ ) and minimum degree in second subset ( $k_2$ ) a new partition is generated where 0 means that vertex does not belong to the core of prespecified  $k_1$  and  $k_2$ , 1 means that vertex belongs to that core.
- \* **2-Mode Review** – Given starting values of  $k_1$  and  $k_2$  the following list is computed:

$k_1$   $k_2$  Rows Cols Comp

where  $k_1$  is minimum degree in the first,  $k_2$  minimum degree in the second subset, *Rows* and *Cols* are number of vertices in first and second subset respectively and *Comp*, number of connected components in network induced by  $k_1$  and  $k_2$ .  $k_1$  and  $k_2$  are incremented until the resulting network is empty.

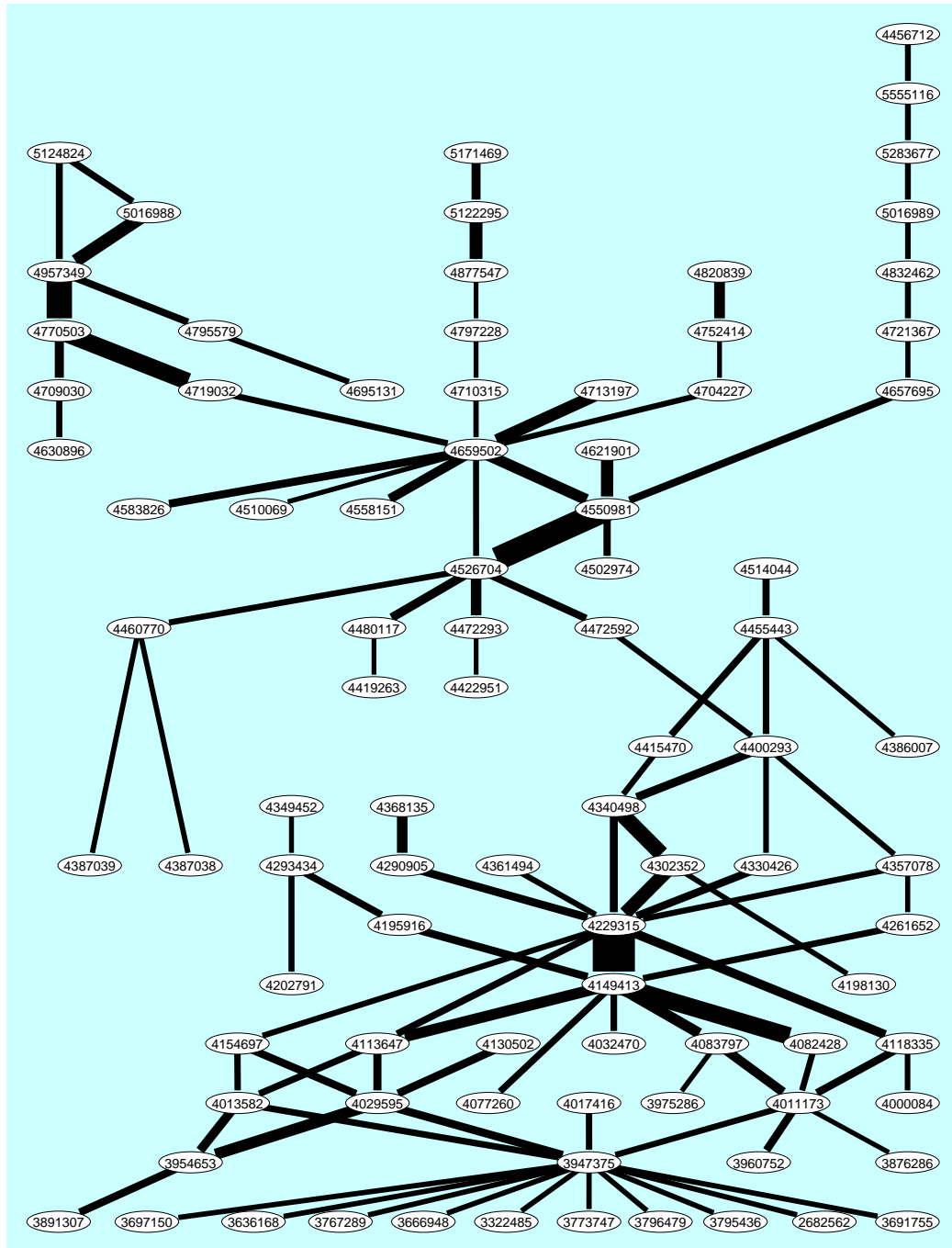


Figure 7: US Patents – Main island 'liquid-crystal display'

- \* **2-Mode Border** – Compute only border values of  $k_1$  and  $k_2$  for a given 2-mode network.
- **Valued Core** – Generalized  $k$ -core: Instead of counting lines (neighbors) use values of lines. sum of lines or maximum value can be used when computing valued core:  
Sum valued core of threshold  $val$  is a subnetwork of given network where the sum of values of lines to (from) the members of the same core is at least  $val$ .  
Max valued core of threshold  $val$  is a subnetwork of given network where the maximum value of all lines to (from) the members of the same core is at least  $val$ .  
Threshold values must be given in advance. Two different ways to determine thresholds:
  - \* **First Threshold and Step** – Select first threshold value and step in which to increase threshold.
  - \* **Selected Thresholds** – Thresholds (increasing numbers) are given using vector.
  - \* **2-Mode** – valued core (according to line values) partition of a 2-mode network. Given minimum valued degree in first ( $k_1$ ) and minimum valued degree in second subset ( $k_2$ ) a new partition is generated where 0 means that vertex does not belong to the valued core of prespecified  $k_1$  and  $k_2$ , 1 means that vertex belongs to that core.
 Additionally (for 1-mode networks), Input, Output or All valued cores can be used.
- **Depth**
  - \* **Acyclic** Partition acyclic network according to depths of vertices.
  - \* **Genealogical** Partition genealogy network according to layers of vertices.
- **$p$ -Cliques** Partition network according to  $p$ -Cliques (partition to clusters where vertices have at least proportion  $p$  (number between 0 and 1) neighbors inside the cluster.
  - \* **Strong** ... for directed network.
  - \* **Weak** ... for undirected network.
- **Vertex Labels** – Partition vertices with same labels to the same class numbers (for molecule).
- **Vertex Shapes** – Partition vertices with same shapes (ellipse, box, diamond) to the same class numbers (used in genealogy to show gender).

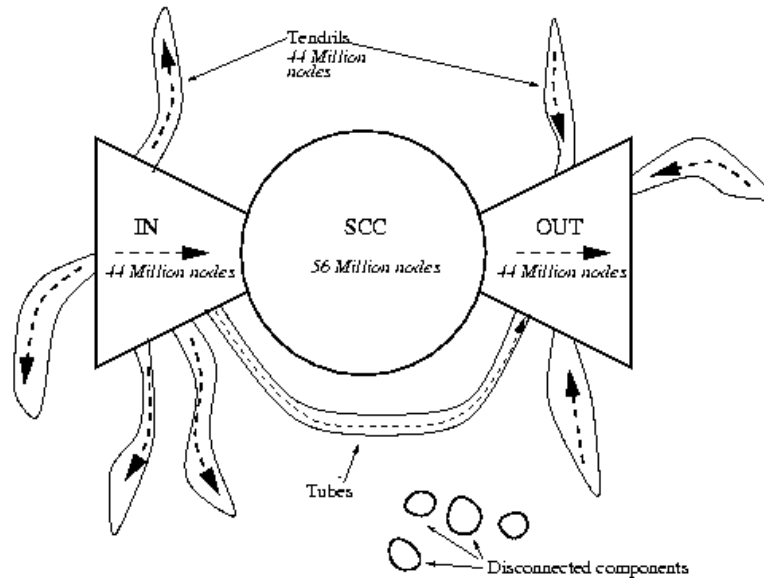


Figure 8: Bow-tie – Graph structure in the web [20]

- **Islands** – Partition vertices of network with values on lines (weights) to cohesive clusters (weights inside clusters must be larger than weights to neighborhood): the height of vertex (vector) is defined as the maximum weight of the neighbor lines. Two options:

- \* **Line Weights**
- \* **Line Weights [Simple]**

New network with only lines constituting islands can be generated if **Generate Network with Islands** is checked.

- **Bow-Tie** – Partition vertices of directed network (graph structure of the web) to the following classes: 1 – LSCC, 2 – IN, 3 – OUT, 4 – TUBES, 5 – TENDRILS, 0 – OTHERS.
- **2-Mode** – Partition of vertices of a 2-mode network into two subsets.

- **Components**

- **Strong** – Strong Components of selected network.
- **Strong-Periodic** – Strong Periodic Components of selected network - strongly connected components are further divided according to periods.



- **Weak** – Weak Components of selected network.
- **Bi-Components** – Biconnected Components of selected network. Articulation points belong to several classes, so the result cannot be stored in partition – biconnected components are stored in hierarchy! Minimal number of vertices in components can be selected. Additionally, partition containing articulation points is produced: number of biconnected components to which each vertex belongs is given. Partition containing vertices belonging to exactly one bicomponent, vertices outside bicomponents and articulation points is also produced: vertices outside bicomponents get class zero, each bicomponent is numbered consecutively (from 1 to number of bicomponents) and articulation points get class number 9999998.

- **Hierarchical Decomposition**

- **Clustering\*** – Hierarchical clustering procedure. Input is dissimilarity network (matrix), which can be obtained using *Operations/Dissimilarity/Network based* or read from input file.
  - \* **Run** – Hierarchical clustering procedure. Result is hierarchy with nested clusters and dendrogram in EPS.
  - \* **Options** – Select method for hierarchical clustering procedure (general, minimum, maximum, average, ward, squared ward).
- **Symmetric-Acyclic** – Symmetric-Acyclic decomposition of network. Result is hierarchy with nested clusters [27].
- **Clustering with Relational Constraint** – Hierarchical clustering with relational constraint procedure. See: Ferligoj A., Batagelj V. (1983): Some types of clustering with relational constraints. *Psychometrika*, **48(4)**, 541-552. Only dissimilarities among vertices that are linked are taken into account what enables to find clusterings very fast also for large networks. Input is network with dissimilarities, which can be obtained using *Operations/Dissimilarity/Network* or *Vector based* or read from input file.
  - \* **Run** – Results are: a partition representing tree: fathers of nodes; and two vectors: describing heights of nodes and number of vertices in subtree respectively. If network has  $n$  vertices then obtained partitions and vectors have dimension  $2*n-1$ . Note that this objects are not compatible with original network, you must use *Make Partition* to get compatible results.

- \* **Make Partition** – From obtained partition representing tree generate partition compatible with original network
  - **using Threshold determined by Vector** – From obtained partition representing tree and one of the two vectors (all have dimension  $2*n-1$ ) generate partition compatible with original network by giving threshold value.
  - **with selected Size of Clusters** – From obtained partition representing tree and given number of vertices in clusters generate partition compatible with original network.
- \* **Extract Subtree as Hierarchy** – Extract subtree from obtained Partition by giving the root as Pajek Hierarchy.
- \* **Options** – Select method for hierarchical clustering with relational constraint (*minimum*, *maximum*, or *average*) and strategy (*strict*, *leader*, or *tolerant*).

- **Numbering**

- **Depth First** – Depth first numbering of selected network...
  - \* **Strong** ... taking directions of arcs into account.
  - \* **Weak** ... forget directions (or undirected network).
- **Breadth First** – Breadth first numbering of selected network...
  - \* **Strong** ... taking directions of arcs into account.
  - \* **Weak** ... forget directions (or undirected network).
- **Reverse Cuthill-McKee** – RCM numbering. [See paper.](#)
- **Core + Degree** – Numbering in decreasing order according to all core partition. Within the same core number vertices are ordered in decreasing order according to number of neighbors which have the same or higher core number.

- **Citation Weights** – If a network represents citation network, weights of lines (citations) and vertices (articles) can be computed. Results are:

- Network with values on lines representing importance of citations.
- Binary partition with vertices on the main path.
- Network containing only main path.
- Vector with importance of vertices (articles).

Different methods of assigning weights [37]:

- **Search Path Count (SPC)** – method. Compute from Source to Sink.
- **Search Path Link Count (SPLC)** – method. Each vertex is considered as Source.
- **Search Path Node Pair (SPNP)** – method.

Weights can also be normalized (using flow or maximum value) or logged.

- ***k*-neighbors** – Select all vertices
  - **Input** ...from which we can reach selected vertex in at most *k*-steps.
  - **Output** ...that can be reached from selected vertex in at most *k*-steps.
  - **All** ...Input + Output (forget direction of lines)  
Result is partition where vertices are in class numbers equal to the distance from given vertex, vertices that cannot be reached from selected vertex are in class number 9999998. After you have a partition you can extract subnetwork.
  - **From Clusters** – Compute selected distances according to each vertex in Cluster. Results consist of so many partitions as is the number of vertices in cluster. Instead of storing results in partitions they can be stored in vectors as well.
- **Paths between 2 vertices**
  - **One Shortest** – Find the shortest path between two vertices. Result is new network. Values on lines can be taken into account (if they present distances between vertices) or not (graph theoretical distance). The latter possibility is faster.
  - **All Shortest** – Find all shortest paths between two vertices. Result is new network. Values on lines can be taken into account (if they present distances between vertices) or not (graph theoretical distance). The latter possibility is faster.
  - **Walks with Limited Length** – Find all walks between two vertices with limited maximum length.
  - **Diameter** – Find diameter – the length of the longest shortest path in network and corresponding two vertices. Full search is performed, so the operation may be slow for very large networks (number of vertices larger than 2000).
  - **Geodesics Matrices\*** – Compute the shortest path length matrix and the geodesics count matrix (*for small networks only!*).

- **Distribution of Distances** – Compute distribution of lengths of the shortest paths and average path length among all reachable pairs of vertices in network.
  - \* **From All Vertices** – Take all vertices as starting points.
  - \* **From Vertices in Cluster** – Only distances from vertices selected by Cluster are computed.
- **Critical Path Method (CPM)** – Find the critical path in acyclic network – result is new network containing the critical path. Algorithm can be used in the area of project planning but also for analysing acyclic graphs. Additional networks containing total and free delay times of activities are generated. Two vectors (partitions) are generated, too: First containing the earliest possible times of coming into given states and the second containing the latest feasible times of coming into given states.
- **Maximum Flow** among vertices.
  - **Selected Pair** – Find maximum flow between selected two vertices (algorithm looks for paths to be saturated and among them it always selects the shortest path). Algorithm can be used in the technical area (actual flow, values on lines mean capacities) or for analysing graphs (if all values are 1). Result is a new network, containing the two vertices and lines contributing to maximum flow between them.
  - **Pairs in Cluster** – Find maximum flow among vertices determined by cluster. Result is a new network, where a value on line means maximum flow between corresponding two vertices. Algorithm is slow: Use it on smaller networks or clusters with limited number of vertices only!
- **Vector** – Get vector from network
  - **Centrality** – Result is a vector containing selected centrality measure of each vertex and centralisation index of the whole network [56, p. 169-219].
    - \* **Closeness** centrality (Sabidussi).
      1. **Input** – centrality of each vertex according to distances of other vertices to selected vertex.
      2. **Output** – centrality of each vertex according to distances of selected vertex to all other vertices.
      3. **All** – forget direction of lines – consider network as undirected.

- \* **Betweenness** centrality (Freeman).
- **Get Loops** – store values of loops to vector.
- **Get Coordinate** – x, y, or z coordinate of network. You can also get all coordinates at once - possibility to have more than 3 coordinates, coordinates must contain character . (dot).
- **Important Vertices** – Find important vertices in directed network (e.g. web pages, scientific citations) or 2-mode network. Result are vectors with weights and partition with selected number of important vertices.
  - \* **1-Mode: Hubs-Authorities** – In directed networks we can usually identify two types of important vertices: *hubs* and *authorities* [39]. A vertex is a good hub, if it points to many good authorities, and it is a good authority, if it is pointed to by many good hubs. In obtained partition value 1 means, that the vertex is a good authority, value 2 means, that the vertex is a good authority and a good hub, and value 3 means, that the vertex is a good hub.
  - \* **2-Mode: Important Vertices** – Generalization of algorithm for 2-mode networks – find important vertices from first and second subset.
- **Structural Holes** – Burt's measure of constraint (structural holes) [21, page 54-55]. Results are:

- \* network  $p_{ij}$ : the proportion of the value of  $i$ 's relation(s) with  $j$  compared to the total value of all relations of  $i$ . where  $a_{ij}$  is the value of the line from  $i$  to  $j$

$$p_{ij} = \frac{a_{ij} + a_{ji}}{\sum_k (a_{ik} + a_{ki})}$$

- \* network containing dyadic constraint  $c_{ij}$  – the constraint of absent primary holes around  $j$  on  $i$ : Explanation: Contact  $j$  constrains your  $i$ 's entrepreneurial opportunities to the extent that:
  - (a) you've made a large investment of time and energy to reach  $j$ , and
  - (b)  $j$  is surrounded by few structural holes with which you could negotiate to get a favorable return on the investment.

$$c_{ij} = (p_{ij} + \sum_{k, k \neq i, k \neq j} p_{ik} p_{kj})^2$$

- \* vector containing aggregate constraint  $C_i$ :  $C_i = \sum_j c_{ij}$ ,  
 $C_i = 1$  for isolated vertices.

- **Clustering Coefficients** – Compute different inherent tendency coefficients in undirected network:

Let  $\text{deg}(v)$  denotes degree of vertex  $v$ ,  $|E(G_1(v))|$  number of lines among vertices in 1-neighborhood of vertex  $v$ ,  $\text{MaxDeg}$  maximum degree of vertex in a network, and  $|E(G_2(v))|$ , number of lines among vertices in 1 and 2-neighborhood of vertex  $v$ .

- \*  $CC_1$  – coefficients considering only 1-neighborhood:

$$CC_1(v) = \frac{2|E(G_1(v))|}{\text{deg}(v) \cdot (\text{deg}(v) - 1)} \quad CC'_1(v) = \frac{\text{deg}(v)}{\text{MaxDeg}} CC_1(v)$$

- \*  $CC_2$  – coefficients considering 2-neighborhood

$$CC_2(v) = \frac{|E(G_1(v))|}{|E(G_2(v))|} \quad CC'_2(v) = \frac{\text{deg}(v)}{\text{MaxDeg}} CC_2(v)$$

If  $\text{deg}(v) \leq 1$  all coefficients for vertex  $v$  are 0.

- **Summing up Values of Lines** – Sum values of all incoming, outgoing or all lines connected to selected vertex.
- **Min of Values of Lines** – Find minimum value of incoming, outgoing or all lines connected to selected vertex.
- **Max of Values of Lines** – Find maximum value of incoming, outgoing or all lines connected to selected vertex.
- **Centers** – Find centers in a graph using 'robbery' algorithm: vertices that have higher degrees (are stronger) than their neighbors steal from them:
  - \* at the beginning give to vertices initial strength according to their degrees, or start with value 1
  - \* when 'weak' vertex is found, neighbors steal from it according to their strengths, or they steal the same amount
- **PCore** – generalized cores.
  - \* **Degree** – ordinary cores.
  - \* **Sum** – taking values of lines into account (sum of values of lines inside pcore).
  - \* **Max** – taking values of lines into account (max of values of lines inside pcore).
- **Count** - how many times each line belongs to predefined rings

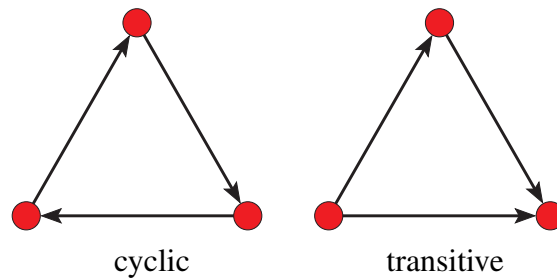


Figure 9: Lines belonging to cyclic and transitive (shortcut) 3-rings

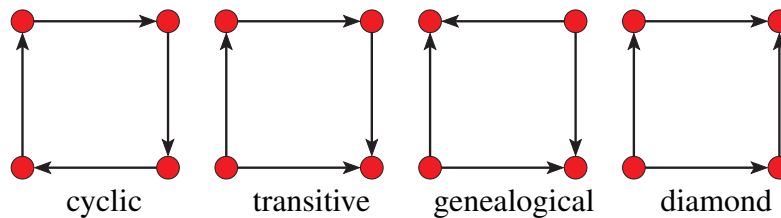


Figure 10: Types of directed 4-rings on arcs

- **3-Rings** – For each line count number of 3-rings to which the line belongs.
  - \* **Undirected** – for undirected networks – count undirected 3-rings.
  - \* **Directed** – for directed networks – count **cyclic**, **transitive**, or all 3-rings, or count how many times each line is a transitive shortcut (see Figure 9).
- **4-Rings** – For each line count number of 4-rings to which the line belongs.
  - \* **Undirected** – for undirected networks – count undirected 4-rings.
  - \* **Directed** – for directed networks – count **cyclic**, **diamonds**, **genealogical**, **transitive**, or all 4-rings, or count how many times each line is a transitive shortcut (see Figure 10).

### 3.3 Nets

Operations on two networks.

- **Union of lines** – Fuse selected networks. Result is a multiple relations network. If you want to get union of networks, multiple lines must still

be deleted. Networks must match in dimension or: If one network has  $m$  vertices and other  $n$  vertices and  $m < n$  then in network with  $n$  vertices first  $m$  vertices must match with vertices in network with  $m$  vertices.

- **Cross-Intersection** – Intersection of selected networks. Networks must match in dimension or: If one network has  $m$  vertices and other  $n$  vertices and  $m < n$  then in network with  $n$  vertices first  $m$  vertices must match with vertices in network with  $m$  vertices.
- **Intersection** – Intersection of selected networks where relation numbers are taken into account.
- **Cross-Difference** – Difference of selected networks.
- **Difference** – Difference of selected networks where relation numbers are taken into account.
- **Union of vertices** – Add the second network at the end of first network.
- **Fragment (1 in 2)** – Find all instances of fragment (determined by network 1) in network 2.
  - **Find** – Execute command.
  - **Options** Select appropriate model of fragment.
    - \* **Induced** – there should be no additional lines between vertices in instance of fragment to match (stronger condition) otherwise additional lines can be present (weaker).
    - \* **Labeled** – labels must match (e.g. atoms in molecule). Labels are determined by classes (colors) in partition - first partition and second partition must be selected before searching for labeled fragments. First partition determines 'labels' of first network (fragment), second partition determines 'labels' of second (original) network.
    - \* **Check values of lines** – values of lines must match (e.g. in genealogy values represent sex: 1 – man, 2 – woman).
    - \* **Check relation number** – relation numbers must match.
    - \* **Check only cluster** – only fragments are searched. where first vertex is one of the vertices in cluster.
    - \* **Extract subnetwork** – produce additional result: extract subnetwork containing vertices belonging to fragments and corresponding lines.



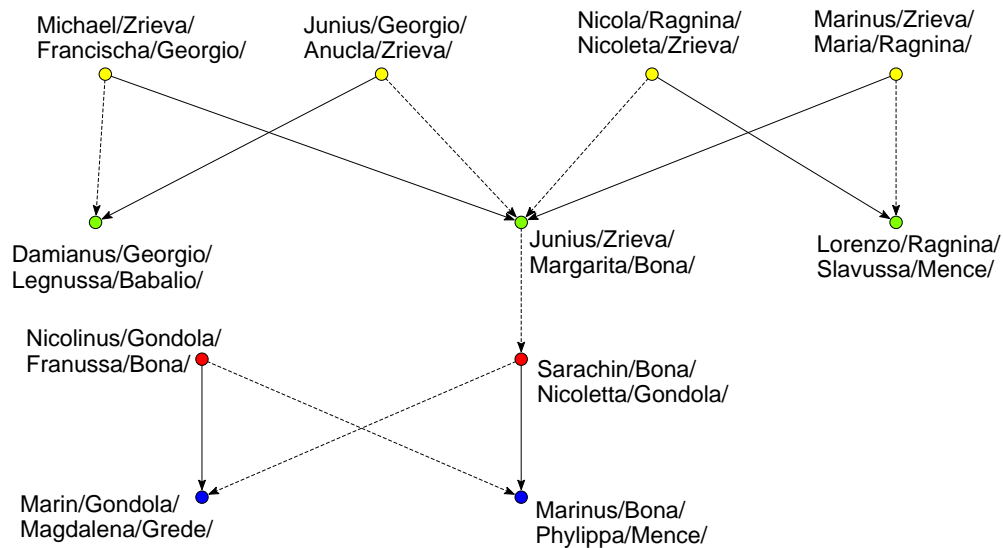


Figure 11: Fragments – Marriages among relatives in Ragusa

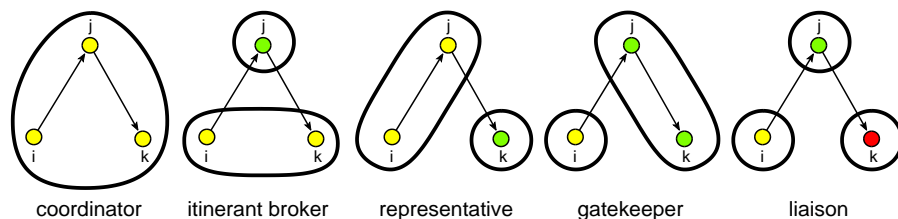
- **Retain all vertices after extracting** – in extracted network the same vertices as in original network are present, only lines which do not belong to any fragment are removed.
- \* **Same vertices determine one fragment at most** – how fragments on the same set of vertices (and different lines) are treated
  - if not checked:* fragments with the same set of vertices are allowed
  - if checked:* fragments with the same set of vertices are not allowed
- \* **Repeating vertices in fragment allowed** – same vertices can appear in fragment more than once (e.g. in cycles).
  - if not checked:* found fragments always have the same number of vertices as original fragment
  - if checked:* some of found fragments can have less vertices than original fragment
- **Multiply First \* Second** - multiply selected 1 or 2 mode networks (that match criteria for multiplication).
- **Shrink coordinates (1 to 2)** - Useful if you shrink network, draw shrunk network separately, and then apply all coordinates to vertices in original network (vertices in same class get the same coordinates). Replace coordinates in network 2 using coordinates of shrunk network 1. Shrinking can be determined using
  - **Partition** or

– Hierarchy

### 3.4 Operations

One network and something else is needed as input.

- **Shrink Network** - Before starting shrinking, select appropriate blockmodel in Options menu. Default is just number of lines between shrunk vertices that must be present in original network, to cause a line in a new network.
  - **Partition** – Shrink network according to selected partition. Vertices in class 0 are (by default) left unchanged, others are shrunk. Results are shrunken network and shrunken partition.
  - **Hierarchy** – Shrink network according to selected hierarchy. Nodes in hierarchy that are *Closed* are shrunk to new vertex. *Cut* nodes are shrunk to virtual vertex. *Border* nodes are not shrunk, but they are not visible. Vertices belonging to other nodes are left unchanged. Type of shrinking (blockmodel) can be selected in Options menu.
- **Extract from Network**
  - **Partition** – Extract sub-network according to selected partition (extract range of classes from partition). Extracted partition is produced as additional result.
  - **Cluster** – Extract sub-network according to selected cluster.
  - **2-Mode Network** – Extract 2-mode network from 1-mode network: first and second mode are determined by given set of clusters in partition.
  - **to GEDCOM** – Extract sub-genealogy according to selected partition (weakly connected component) to new GEDCOM file (genealogy must be read as Ore graph).
- **Brokerage Roles** - For each vertex  $j$  count five brokerage roles (*coordinator*, *itinerant broker*, *representative*, *gatekeeper* and *liaison*) according to given partition.



- **Dissimilarity\***

- **Network based** – Compute selected dissimilarity matrix ( $d_1, d_2, d_3$  or  $d_4$ ) among vertices in cluster according to number of common neighbors. *Corrected Euclidean-like*  $d_5$  and *Manhattan-like*  $d_6$  dissimilarities can be computed as well [10]. The obtained matrix can be used further for hierarchical clustering procedure.

You can include vertex  $v$  to its own neighborhood or not and display in report window only upper triangle / undirected or complete matrix /directed (if number of vertices is low).

$N_v$  is a set of input, output or all neighbors of vertex  $v$ ; + stands for symmetric sum,  $\cup$  stands for set union and  $\setminus$  stands for set difference;  $|$  stands for set cardinality; 1st maxdegree and 2nd maxdegree are the largest degree and the second largest degree in network, respectively.

$$d_1(u, v) = \frac{|N_u + N_v|}{\text{1st maxdegree} + \text{2nd maxdegree}}$$

$$d_2(u, v) = \frac{|N_u + N_v|}{|N_u \cup N_v|}$$

$$d_3(u, v) = \frac{|N_u + N_v|}{|N_u| + |N_v|}$$

$$d_4(u, v) = \frac{\max(|N_u \setminus N_v|, |N_v \setminus N_u|)}{\max(|N_u|, |N_v|)}$$

$$d_5(u, v) = \sqrt{\sum_{\substack{s=1 \\ s \neq u, v}}^n ((q_{us} - q_{vs})^2 + (q_{su} - q_{sv})^2) + p \cdot ((q_{uu} - q_{vv})^2 + (q_{uv} - q_{vu})^2)}$$

$$d_6(u, v) = \sum_{\substack{s=1 \\ s \neq u, v}}^n (|q_{us} - q_{vs}| + |q_{su} - q_{sv}|) + p \cdot (|q_{uu} - q_{vv}| + |q_{uv} - q_{vu}|)$$

Dissimilarities  $d_5$  and  $d_6$  are based on some matrix  $\mathbf{Q} = [q_{uv}]$  on vertices – for example on adjacency matrix or on distance matrix. The parameter  $p$  is usually set to value 1 or 2. In the case  $N_u = N_v = 0$  we set all dissimilarities  $d_1 - d_4$  to 1.

If **Among all linked Vertices only** is checked dissimilarities are computed as line values of given network.

- **Vector based – Euclidean, Manhattan, Canberra, or (1-Cosine)/2** dissimilarities among Vectors determined by Cluster are computed as line values of given network.

- **Vector** – Operations on network and vector.
  - **Network \* Vector** – Ordinary multiplication of matrix (network) by vector. Result is a new vector.
  - **Vector # Network** – Result is a new network:
    - \* **Input** – Multiplying incoming arcs in network by corresponding vector values - multiplying  $i$ -th column of matrix by  $i$ -th component of vector.
    - \* **Output** – Multiplying outgoing arcs in network by corresponding vector values - multiplying  $i$ -th row of matrix by  $i$ -th component of vector.
  - **Harmonic Function** – See Bollobas [19, page 328].  
 Let  $(G, a)$  be a connected weighted graph, with weight function  $a(x, y)$ , and let  $S$  is subset of vertices  $V(G)$ . A function  $f : V(G) \rightarrow \mathbb{R}$  is said to be harmonic on  $(G, a)$ , with boundary  $S$ , if

$$f(x) = \frac{1}{A(x)} \sum_y (a(x, y)f(y)), \quad \forall x \in V(G) \setminus S$$

$$A(x) = \sum_y a(x, y)$$

Implementation in **Pajek**:

- \* function  $f$  is determined by vector
- \* weight function  $a(x, y)$  is given by (valued) network
- \* subset  $S$  is determined by partition – vertices in class 1 are in subset  $S$  (fixed vertices), other vertices are in  $V(G) \setminus S$
- \* additionally, permutation determines the order of vertices in computations.

In **Pajek** you can compute the harmonic function once or iteratively - as long as difference between successive functions become small enough. Components of vector that represents function  $f$  can be modified immediately when they are computed or only at the end of each iteration (after all components are computed). Procedure can be run according to:

- \* **Input** – neighbors
- \* **Output** – neighbors
- \* **All** – neighbors

- **Summing up neighbors** – For each vertex compute the sum of class numbers of its neighbors according to
  - \* **Input** – neighbors
  - \* **Output** – neighbors
  - \* **All** – neighbors
- **Min of neighbors** – For each vertex compute the minimum class number of its neighbors according to
  - \* **Input** – neighbors
  - \* **Output** – neighbors
  - \* **All** – neighbors
- **Max of neighbors** – For each vertex compute the maximum class number of its neighbors according to
  - \* **Input** – neighbors
  - \* **Output** – neighbors
  - \* **All** – neighbors
- **Put Loops** – put vector values as loops (arcs or edges) in current network.
- **Put Coordinate** – put vector as x, y, or z coordinate, or put it as polar radius or polar angle of vertices in network layout.
- **Diffusion Partition** – Compute diffusion partition according to thresholds given in vector. Vertices in selected cluster are considered to adopt in time 1.
- **Islands** – Partition vertices to cohesive clusters according to weights of vertices determined by a vector.
  - \* **Vertex Weights** – Vertex island is a cluster of vertices of given network with weighted vertices where the weights of the vertices on the island are larger than the weights of the vertices in the neighborhood. The weights are also called heights.
  - \* **Vertex Weights [Simple]** – Simple vertex island is vertex island with only one top.
- **Transform** – Transformations of network according to Partition, Cluster and/or Vector.
  - **Remove Lines** – Removing lines according to partition.
    - \* **Inside Clusters** – Remove all lines with incident vertices in the same (selected) cluster(s).

- \* **Between Clusters** – Remove all lines with incident vertices in different clusters.
  - \* **Between Two Clusters**
    1. **Arcs** – Remove all arcs pointing from first to second cluster.
    2. **Edges** – Remove all edges between the selected two clusters.
  - \* **Inside Clusters with value**
    1. **lower than Vector value** – Remove all lines inside clusters (determined by a Partition) with value lower than the value specified in a Vector.
    2. **higher than Vector value** – Remove all lines inside clusters (determined by a Partition) with value higher than the value specified in a Vector.

Dimension of a Vector must be equal to the highest cluster number in a Partition.
  - **Add** – some elements to network
    - \* **Arcs from Vertex to Cluster** – add arcs from selected vertex to all vertices in Cluster.
    - \* **Arcs from Cluster to Vertex** – add arcs from all vertices in Cluster to selected vertex.
    - \* **Time Intervals determined by Partitions** – change network to temporal network using two partitions: first partition determines initial time point, second determines terminal time point of each vertex.
  - **Direction** – Convert to directed network where all arcs are pointing from
    - \* **Lower->Higher** class number.
    - \* **Higher->Lower** class number.

Lines inside classes may be deleted or not.
  - **Vector(s) -> Line Values** – Replace line values with result of selected operation (sum, difference, multiplication, division) on vector(s) values in corresponding terminal and initial vertices.
- **Reorder**
    - **Network** – Reorder vertices in network according to selected permutation.
    - **Partition** – Reorder vertices in partition according to selected permutation.

- **Vector** – Reorder vertices in vector according to selected permutation.
- **Count neighbor Colors** – For selected network and partition a new partition is generated where for each vertex the frequency of vertices of selected color in the neighborhood is given. Colors to be counted are determined using cluster.
- **Coloring**
  - **Create New** – Sequential coloring of vertices in order determined by permutation. Result depends on selected permutation significantly.
  - **Complete Old** – Complete partial coloring of vertices in order determined by permutation. For example some vertices can be colored by hand, but most of the vertices are still uncolored (in class 0). In this way you can help program to produce better coloring.
- **Balance\*** – Relocation algorithm for partitioning signed graphs (graphs with positive and negative values on lines representing friends and enemies, for example). Given partition is optimized to get as much as possible positive lines inside classes and negative lines between classes. Another algorithm does not distinguish between diagonal and off-diagonal blocks: each block can be positive, negative, or null. If number of repetitions is higher than 1, initial partitions into given number of classes are chosen randomly for every repetition separately. If program finds several optimal solutions, all are reported. For more details about algorithm see Doreian and Mrvar [26].

Option can be used for two mode signed graphs as well: input is two mode partition. In this case algorithm tries to find as 'clear' as possible positive, negative, and null blocks.

By setting penalty for small null blocks to some nonzero value, we try to get null blocks as large as possible.
- **Blockmodeling\*** – Generalized blockmodeling of 1-mode and 2-mode networks [6, 29]. For details see Section 7 on page 78. Descriptions of models are stored on MDL files. See also block types on page 48.
  - **Random Start** – Start the optimization with random partition(s).
  - **Optimize Partition** – Show the criterion function for selected partition and optimize it.
  - **Restricted Options** – Show only selected part of options (sufficient for most users) or all options.

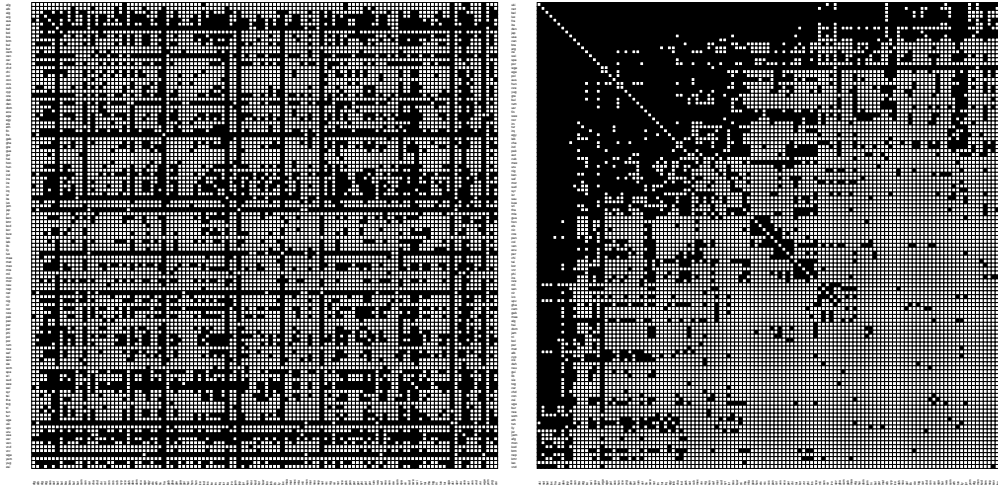


Figure 12: World trade. Orderings: alphabetical and determined by clustering

- **Short Report** – Show only main results of optimization in Report window (sufficient for most users) or detailed, long report.
- **Genetic Structure** – Compute genetic structure of given acyclic network according to given partition (of minimal vertices). As result we get as many vectors as is different clusters in partition, and the dominant gene partition.
- **Permutation\*** – Improve given permutation according to network.
  - **Travelling Salesman** – Can be applied to dissimilarity matrix, or modified matrix representing network (fill diagonal and change 0 in the matrix with some large numbers):
    - \* **Run** – Run 3-OPT algorithm for solving Travelling Salesman Problem.
    - \* **Options** – Put selected value on diagonal, add some artificial vertices, and incident lines with large values, change value 0 with selected (large) value.
  - **Seriaton** – Starting with network and (random) permutation improve the permutation using seriation algorithm from Murtagh [45, page 11-16].
    - \* **1-Mode** – for ordinary (1-Mode) networks
    - \* **2-Mode** – for 2-Mode networks



- **Clumping** – Starting with network and (random) permutation improve the permutation using clumping algorithm from Murtagh [45, page 11-16].
  - \* **1-Mode** – for ordinary (1-Mode) networks
  - \* **2-Mode** – for 2-Mode networks
- **R-Enumeration** – Starting with network and (random) permutation find such permutation that enumeration of neighbor vertices are as close to each other as possible.
- **Functional Composition** – Let  $f$  be a partition or a permutation and  $g$  a partition, a permutation, or a vector. The result is new partition, permutation or vector  $r$  defined in the following way:  $r[v] = (f * g)[v] = g[f[v]]$ .
- **Expand Partition**
  - **Greedy Partition** – Put vertices with unknown class number (0) in the same class as selected vertices in partition if
    - \* **Input** ...we can reach selected vertices in at most  $k$ -steps.
    - \* **Output** ...we can come to vertices from selected vertices in at most  $k$ -steps.
    - \* **All** ...Input + Output (forget direction of lines)
 Classes are joined if one vertex should belong to more classes.
  - **Influence Partition** – Put every vertex with unknown class number (0) in given partition in the same class as is the class of the closest vertex. If several vertices with known class number have the same distance, the highest value is used.
  - **Make Multiple Relations Network** – Transform network to a multiple relation network using a partition: if both endvertices of a line belong to the same class in partition the multiple relations tag will be equal to the class number of endvertices, otherwise it will be 0.
- **Expand Reduction** – Restore original network from reduced network (hierarchical reduction!) and appropriate hierarchy (result is always undirected network).
- **Identify** – Identify (reorder and/or join some units).
- **Petri** – Execute Petri net according to starting marking of places determined by partition. Number of places in network is equal to dimension of partition. Places must be defined first ( $1..m$ ) then transitions ( $m + 1..n$ ). What to do if more than one transition can fire? Two possibilities:

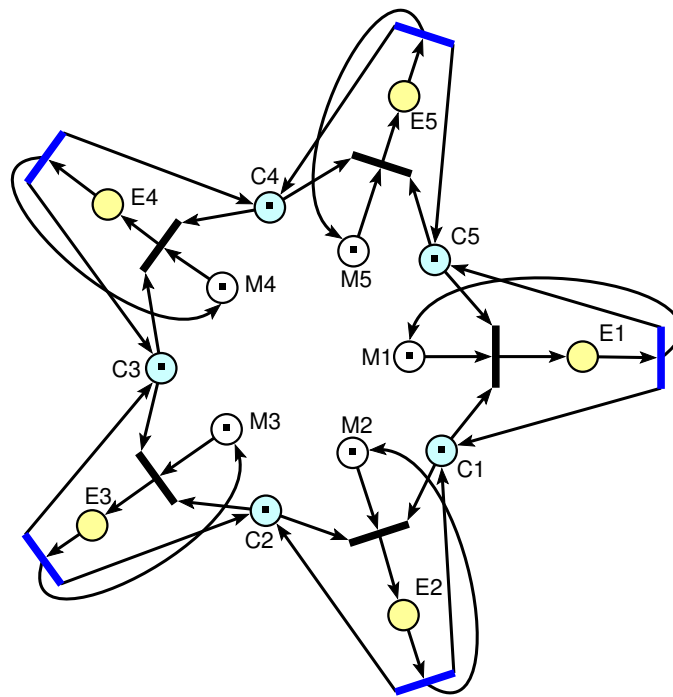


Figure 13: Petri net

- **Random** – Transition is chosen randomly.
- **Complete** – Complete tree of all possible transitions is built - result is hierarchy. You can choose the maximum depth of the tree, or execute Petri net as long as possible.

Try for example `petri2` from the book of Peterson [48, page 21] or `petri52` (see Figure 13) `data`.

- **Refine Partition** Refine partition according to selected network (reachability).
  - **Strong** ... for directed network.
  - **Weak** ... for undirected network.
- **Leader Partition** – find clusters of vertices of network inside layers.

### 3.5 Partition

Only Partition is needed as input.

- **Create Constant Partition** – Create constant partition of selected dimension. Default dimension is the size of selected network (if there is one in memory).
- **Create Random Partition** – Create random one or two mode partition.
- **Binarize** – Make binary (0-1) partition from selected partition.
- **Fuse Clusters** – Fuse selected cluster numbers to a new cluster.
- **Canonical Partition** – Transform partition to its canonical (unique) form (vertex 1 is always in class 1, the next vertex with smallest number that is not in the same class as vertex 1 is in class 2...).
- **Canonical Partition [Decreasing frequencies]** – Transform partition to its canonical (unique) form (in class 1 the old class with the highest frequency will be set, in class 2 the old class with the second highest frequency...).
- **Make Network** – Generate network from partition.
  - **Random Network** – Generate random network where degrees of vertices are determined using partition.
    - \* **Undirected** – partition gives degrees of vertices in undirected network.

- \* **Input** – partition gives input degrees of vertices.
- \* **Output** – partition gives output degrees of vertices.
- **2-Mode Network** – Generate 2-mode network: first set consists of vertices ( $v_1 \dots v_n$ ), second set consists of clusters ( $c_0 \dots c_m$ ). If vertex  $i$  is in cluster  $j$  the line from  $v_i$  to  $c_j$  is generated. If option *Existing Clusters only* is selected only clusters containing at least one vertex are generated as vertices in the second set.
- **Make Permutation** – Make permutation from selected partition. (first all vertices with the lowest class number, ...)
- **Make Cluster** – Transform partition to cluster.
- **Make Hierarchy** – Transform partition to hierarchy (nested or not).
- **Make Vector** – Transform partition to vector ( $V[i] := C[i]$ ).
- **Count, Min-Max Vector** – info about cluster frequencies and minimum and maximum vector value according to given partition.

### 3.6 Partitions

Operations on two partitions. Two partitions must be selected before performing operations.

- **Extract second from first** – Extract from first partition vertices that satisfy criterion (are on specified interval) determined by second partition. This operation is useful when we have partition that actually saves some information about vertices (for example gender). When you get (extract) some smaller part of the network (for example vertices that are on distances less than 3 from selected vertex), information about gender would be lost without performing the same operation (extraction) on partition.
- **Add Partitions** – Add two partitions (useful for example when combining Input and Output neighbors in acyclic networks).
- **Min (C1, C2)** – Minimum of two partitions.
- **Max (C1, C2)** – Maximum of two partitions.
- **Fuse Partitions** – Fuse two partitions – add second to the end of the first (useful for 2-mode networks).
- **Expand** – Expand partition to higher (original) dimension.

- **First according to Second (Shrink)** – Expand first partition according to shrinking determined by second partition.
- **Insert First into Second according to Third (Extract)** – The current partition was obtained by extracting selected classes defined by the second partition from the first partition. This sub-partition was modified. Using this operation we can insert this modified sub-partition back to the first partition.
- **Intersection** – of selected partitions.
- **Cover with** – Let  $p$  be a partition,  $b$  a binary partition, and  $c$  selected cluster number. Result is new partition  $q$  determined in the following way:  
**if**  $b(v) = 0$  **then**  $q(v) = p(v)$  **else**  $q(v) = c$ .
- **Make Random Network** – generate random network with input degrees determined by the first and output degrees by the second partition.
- **Info** – Bivariate statistical measures between selected partitions:
  - **Cramer's V, Rajski** – Report contingency table, compute Cramer's V and Rajski coefficients.
  - **Spearman Rank** correlation coefficient.

### 3.7 Vector

Operations using vector.

- **Create Constant Vector** – Create constant vector (vector with all values equal to selected value) of selected dimension. Default dimension is the size of selected network (if there is one in memory).
- **Extract Subvector** – Extract subvector from given vector - criterion is class in the selected partition.
- **Shrink Vector** – Shrink vector values according to clusters of partition to new vector – adjusting vector to shrunken network. When shrinking several values to one value, sum of values, mean, min, max or median value can be used.
- **Make Partition** – Convert vector to partition:
  - **by Intervals** – according to selected dividing numbers in vector vertices get appropriate class numbers. Intervals can be given by:

- \* **First Threshold and Step** – Select first threshold and step in which to increase threshold.
- \* **Selected Thresholds** – Select all thresholds or number of classes (#) in advance.
  - by **Truncating (Abs)** – partition is absolute and truncated vector.
- **Make Permutation** – Convert vector to permutation - sorting permutation.
- **Transform** – Transformations of given vector:
  - **Multiply by** a constant.
  - **Add Constant** to vector values.
  - **Absolute** values of its elements.
  - **Absolute + Sqrt** – square root of its absolute components.
  - **Truncate** – truncated vector.
  - **Exp** – exponential of vector.
  - **Ln** – natural logarithm of vector.
  - **Power** – selected power of vector.
  - **Normalize**
    - \* **Sum** – normalize so that the sum of elements is 1.
    - \* **Max** – normalize so that the maximum element will have value 1.
    - \* **Standardize** – normalize so that arithmetic mean will be 0 and standard deviation 1.

### 3.8 Vectors

Operations on two vectors. Two vectors must be selected before performing operations.

- **Add Vectors** – sum of selected vectors.
- **Subtract Second from First** – difference of selected vectors.
- **Multiply Vectors** – product of selected vectors.
- **Divide First by Second** – division of selected vectors.
- **Linear Regression** – fit the two vectors using linear regression. Results are: regression line, linear estimates of second vector and corresponding errors.

- **Min (V1, V2)** – smaller elements in selected vectors.
- **Max (V1, V2)** – bigger elements in selected vectors.
- **Fuse Vectors** – fusion of vectors.
- **Transform** – two vectors to another two vectors:
  - **Cartesian** → **Polar** – First vector must contain  $x$ -coordinates second  $y$ -coordinates. Results are: vector containing polar radius and vector containing polar angles in degrees.
  - **Polar** → **Cartesian** – First vector must contain polar radius second polar angles in degrees. Results are: vector containing  $x$ -coordinates and vector containing  $y$ -coordinates.

Results can be (de)normalized to enable direct use in Draw window.

- **Info** – Pearson correlation coefficient between selected vectors.

### 3.9 Permutation

Only permutation is needed as input.

- **Identity** – Create identity permutation of selected dimension. Default dimension is the size of selected network (if there is one in memory).
- **Random** – Create random permutation of selected dimension. Default dimension is the size of selected network (if there is one in memory).
- **Random 2-Mode** – Create random permutation of selected dimension and number of vertices in the first subset of 2-mode network. Default dimension is the size of selected network and number of vertices in the first subset (if there is network and corresponding partition in memory).
- **Inverse** – Create inverse permutation of selected permutation.
- **Mirror** – Create mirroring permutation of selected permutation (sort in opposite direction).
- **Make Partition** – Create partition into selected number of clusters from given permutation.
- **Make Vector** – Transform permutation to vector.

### 3.10 Permutations

Operation on two permutations.

- **Fuse Permutations** – Fuse two permutations – add second to the end of the first (useful for 2-mode networks).

### 3.11 Cluster

Only cluster (and partition) is needed as input.

- **Create Empty Cluster** – Create cluster without vertices.
- **Create Complete Cluster** – Create cluster with values 1..n.
- **Make Partition** – Transform cluster to partition.
- **Binarize Partition** – Binarize partition according to cluster - make binary partition of the same dimension as the given partition, vertices that are in cluster numbers determined by the cluster will go to class 1 other to class 0. This allows noncontiguous ranges to be selected (other choices in **Pajek** need contiguous ranges). Note the exception: In this case cluster represents set of cluster numbers and not set of vertices numbers.

### 3.12 Hierarchy

Only hierarchy is needed as input.

- **Extract Cluster** – Extract cluster from hierarchy - the cluster is whole subtree of selected node in hierarchy.
- **Make Network** – Converts hierarchy to network (use it for example to draw hierarchy – drawing by layers). Closed nodes are also taken into account.
- **Make Partition** – Converts hierarchy to partition (according to closed nodes).
- **Make Permutation** – Converts hierarchy to permutation.

### 3.13 Options

- **Read - Write**
  - **Threshold** – Value of line must be higher (absolutely) than the given threshold to generate line between two vertices.



- **Max. vertices to draw** – Maximum number of vertices in network to allow drawing (to prevent long waiting).
- **Read - Save vertices labels?** – Read / Save also labels, coordinates, and other descriptions of vertices or not. If vertices labels are not read (recommended if network is very large and vertices labels are long) they can be imported later from input file using **Net/Transform/Add/Vertex Labels from File**.
- **Save coordinates of vertices?** – Save coordinates of vertices to network file (or not).
- **Save complete vertex description?** – When saving network to output file for each vertex complete description will be written, even if consequent vertices have the same descriptions (e.g. shapes, time intervals...).
- **Check equality of vertex descriptions by reading?** – Enables users to speed up reading large network files according to descriptions of vertices: Check this option to save space when exactly the same descriptions of vertices are repeated often (e.g. shapes of vertices). Uncheck this option to save time when there are several different descriptions of vertices in input file (e.g. time stamps in temporal networks).
- **Check equality of line descriptions by reading?** – Enables users to speed up reading large network files according to descriptions of lines: Check this option to save space when exactly the same descriptions of lines are repeated often (e.g. line pattern Dots/Solid). Uncheck this option to save time when there are several different descriptions of lines in input file (e.g. time stamps in temporal networks).
- **Auto Report?** – Automatically report all text results to file repl.rep.
- **Ore: Different relations for male and female links** – When reading genealogy as Ore graph generate 2 different types of arcs: arc with relation number 1 (also value 1) represents (god)father to child relation, arc with with relation number 2 (also value 2) represents (god)mother to child relation.
- **Ore: Generate Godparent relation** – When reading genealogy as Ore graph generate also godparent relation (relation number 4) or godfather (relation number 4) and godmother (relation number 5) relations.
- **GEDCOM – Pgraph** – Use pgraph format (nodes are couples or individuals) when reading genealogies (D. R. White), otherwise nodes are only individuals.

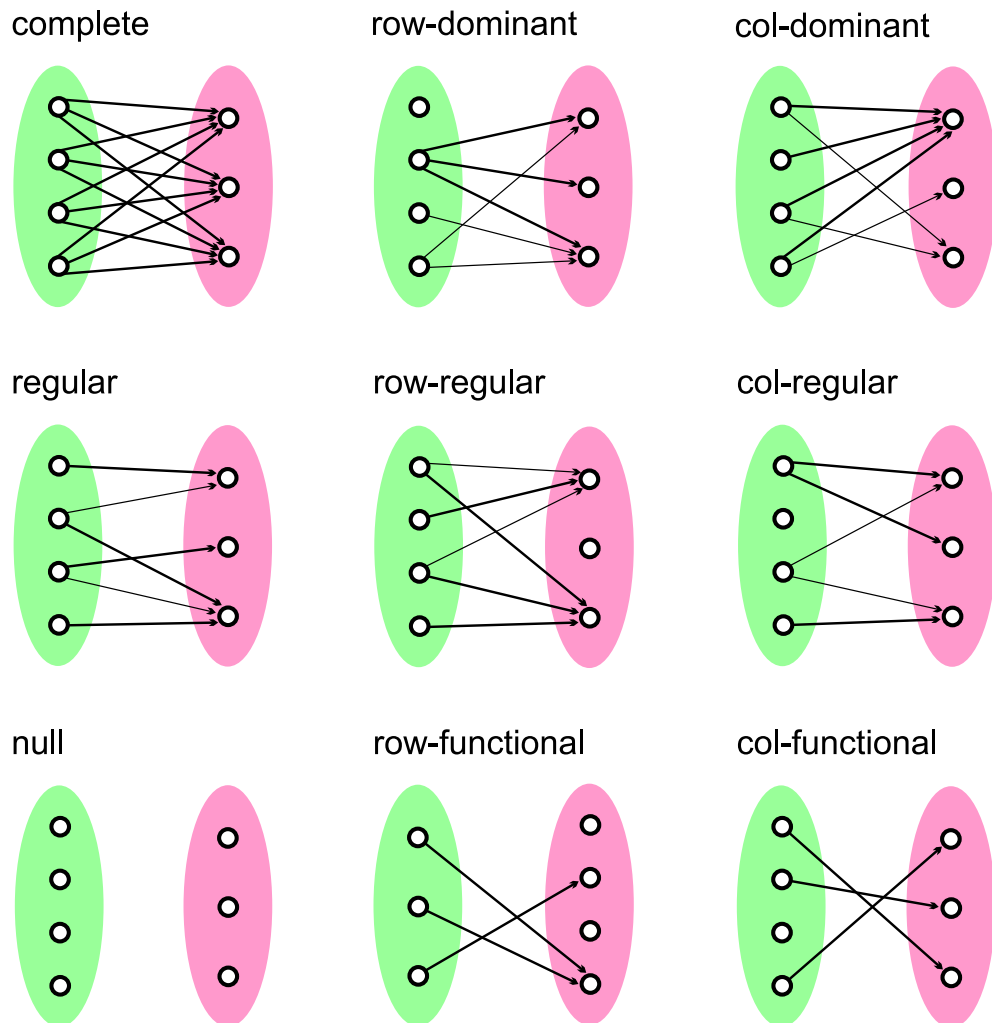


Figure 14: Generalized block types

- **Bipartite Pgraph** – Generate bipartite pgraph that has squares for marriages, triangles and circles for individuals.
- **Pgraph+labels** – Attach also labels of lines to pgraph, when reading GEDCOM file.
- $x / 0$  = Specify the result when dividing nonzero value with 0.
- $0 / 0$  = Specify the result when dividing 0 with 0.
- **Blockmodel** – Select type of blockmodel for shrinking. Possibilities are:
  - 0..Min Number of Links

- 1..Null
- 2..Complete
- 3..Row-Dominant
- 4..Col-Dominant
- 5..Row-Regular
- 6..Col-Regular
- 7..Regular
- 8..Row-Functional
- 9..Col-Functional
- 10..Degree Density

Look in Batagelj [6] and Doreian, Batagelj, Ferligoj [29].

- **Ini File**

- **Load** – Use selected configuration of **Pajek** which is stored in the file (\*.ini).
- **Save** – Save the current configuration of **Pajek** into a file (\*.ini).

- **FontSize** – Size of Font for displays.

### 3.14 Info

- **Network** – Information about network
  - **General** – General information about network
    - \* number of vertices
    - \* number of arcs, edges and loops
    - \* density of lines
    - \* sort lines according to their values (ascending or descending) to find the most/least important lines.
  - **Line Values** – Frequency distribution of line values.
  - **Indices** – Different indices on network (chemical and genealogical).
  - **Triadic Census** – Number of different triads in network. See book of Faust and Wasserman [56] and Figure 15 on page 52.
  - **Multiple Relations** – General information about multiple relations network

- \* number of relations
- \* number of arcs, edges and total number of lines for each relation
- **Vertex Label -> Vertex Number** – Find vertex number by giving (part of) its label, or find vertex label for given vertex number.
- **Partition** – General information about partition. Sort vertices according to their class numbers (ascending or descending) to see the most important vertices. Frequency distribution of class numbers. Average, median and standard deviation of class numbers are also given.
- **Hierarchy** – General information about hierarchy. Operation is possible only if node numbers are integers. It returns number of vertices in nodes of hierarchy (on first level).
- **Vector** – General information about vector: Vertices sorted according to their values, average, median, standard deviation and frequency distribution of vector values into given number of classes (# – number of classes or selected dividing values can be given).
- **Memory** – Available memory. Not very accurate.
- **About** – Information about Pajek version, authors, copyrights...

### 3.15 Tools

- **R**
  - **Send to R** – Call statistical package R [49] with one vector/network, vectors/networks selected by cluster or all currently available vectors and/or networks.
  - **Locate R** – locate position of statistical program R (Rgui.exe or Rterm.exe) on the disk.
- **SPSS**
  - **Send to SPSS** – Call statistical package SPSS with one partition, vector or network, partitions/vectors selected by cluster or all currently available partitions and vectors.
  - **Locate SPSS** – locate position of statistical program SPSS (runsyntx.exe) on the disk.
- **Web Browser** – Select which web browser to open when clicking on vertex with *Shift* and *Right mouse button*.

- **Add Program** – add new executable program with specified parameters to the tools menu.
- **Edit Parameters** – edit parameters of selected external program.
- **Remove Program** – remove selected external program from the tools menu.

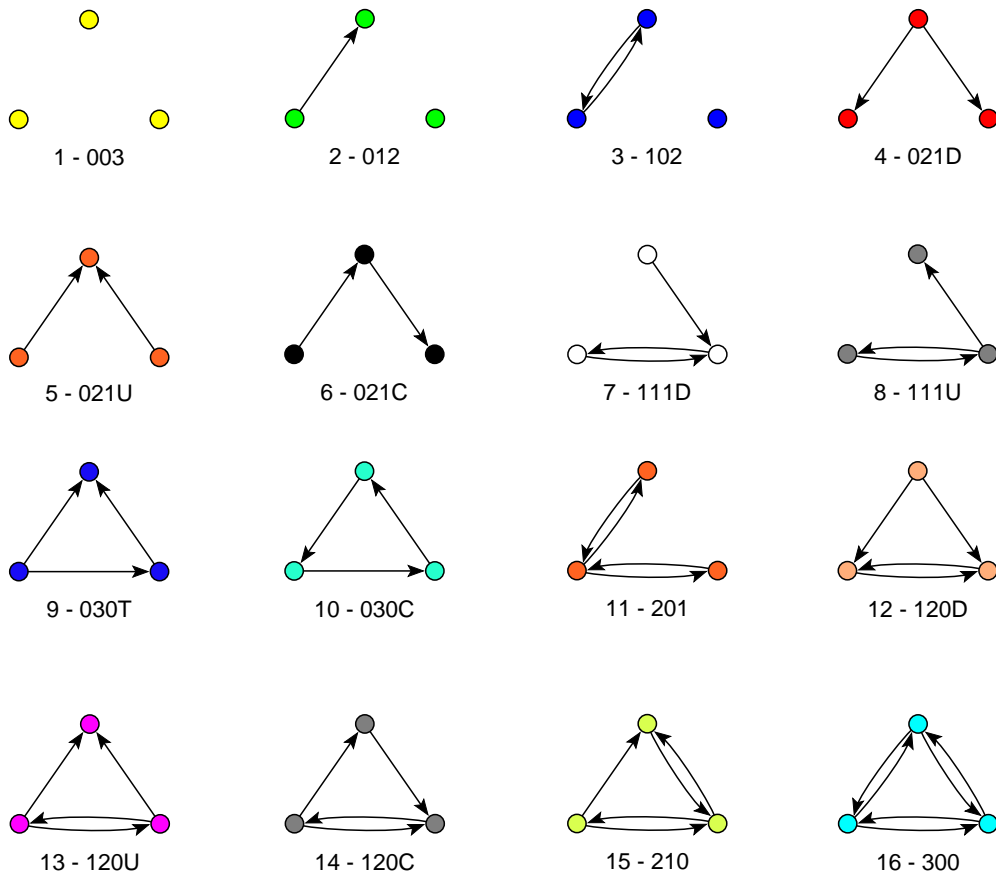


Figure 15: All different triads.

## 4 Draw Window Tools

### 4.1 Main Window Draw Tool

- **Draw** - Draw Network. A new window is open, where a new menu appears. You can edit network by hand (move vertices using left mouse button), select a part of the picture (using right mouse button and select the area), edit lines that belong to selected vertex by clicking on vertex using right mouse button, spin picture using keys X, Y, Z, S, x, y, z, s. Description of Draw window menu:

- **Draw-Partition** – Similar to **Draw**. Colors of vertices represent the classes in selected partition. Additionally you can put selected vertex or selected vertices into given class in partition (classes are shown using different colors) by clicking on middle mouse button (or Shift+left button) (increment class), or together with Alt (or Alt+left button) - decrement class number. In Figure 20 on page 84 you can see which color represents selected class.

Some additional menu items that were already described appeared (you can draw network according to layers from partition and optimise energy using fixed vertices determined using partition). It is also possible to move the selected class (by clicking close to vertex from that partition).

- **Draw-Vector** – Sizes of vertices are determined using selected vector.
- **Draw-2Vectors** – Sizes of vertices are determined using selected two vectors (first for width second for height).
- **Draw-Partition-Vector** – Colors of vertices are determined using selected partition, sizes of vertices are determined using selected vector.
- **Draw-Partition-2Vectors** – Colors of vertices are determined using selected partition, sizes of vertices are determined using selected two vectors (first for width second for height).
- **Draw-SelectAll** – Create null partition and draw network using it.

### 4.2 Layout

Generate layout of the network.

- **Circular** – Position vertices on circle
  1. **Original** – in order determined by the network.

2. **using Permutation** – in order determined by current permutation.
  3. **using Partition** – create separate circles for clusters in selected partition. Center of the circle is determined by the arithmetic mean of positions of vertices in the cluster.
  4. **Random** – in random order.
- **Energy** – Automatic layout generation.
    1. **Kamada-Kawai** – algorithm for automatic layout generation in the plane.
      - (a) **Free** – Every position in the plane is possible.
      - (b) **Fix first and last** – First and last vertex are fixed in opposite corners.
      - (c) **Fix one vertex in the middle** - Select vertex which will be fixed in the middle of the picture.
      - (d) **Selected group only** – Only selected part of the picture is taking into account during optimisation.
      - (e) **Fix selected vertices** - Selected vertices (from partition) are fixed on given positions). This item is visible only if Draw partition is active.
    2. **Fruchterman Reingold** – another algorithm for automatic layout generation (faster than Kamada-Kawai).
      - (a) **2D** – optimisation in plane.
      - (b) **3D** – optimisation in space.
      - (c) **Factor** – Input factor for optimal distance among vertices when using Fruchterman Reingold optimisation.
    3. **Starting positions** – for energy drawing (random, circular, given positions on plane xy, given z coordinates).
  - **EigenValues** – Drawing using eigenvalues/eigenvectors (Lanczos algorithm). Values of lines can be taken into account or not.
    1. **1 1 1** – Select 2 or 3 eigenvalues and algorithm will compute corresponding eigenvectors. Eigenvalues may be multiple so there are many possibilities. Some examples
      - (a) **1 1 1** – compute 3 eigenvectors that correspond to the first eigenvalue
      - (b) **1 1 2** – compute 2 eigenvectors that correspond to the first eigenvalue and 1 that correspond to the second



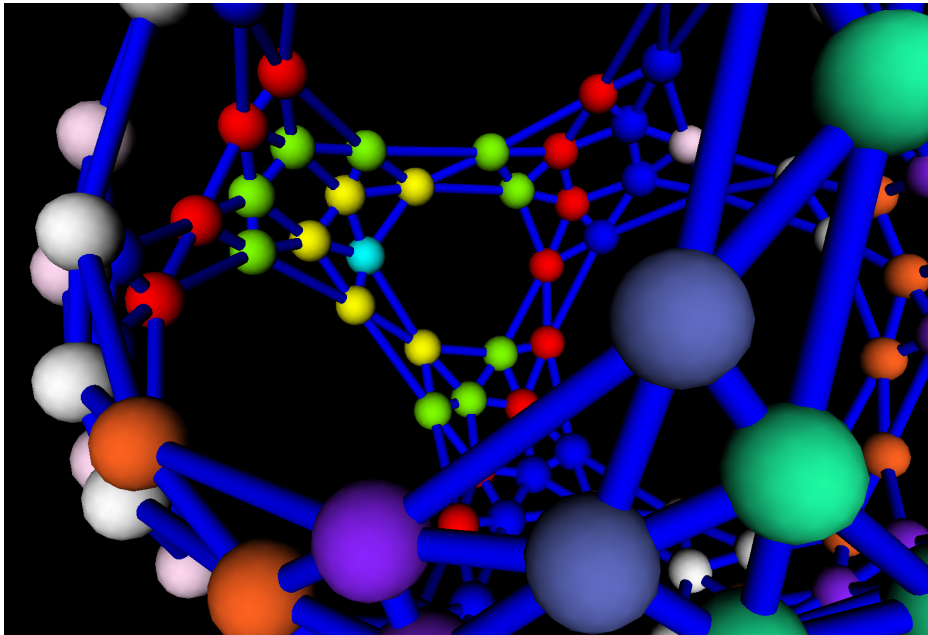


Figure 16: VRML display of layout determined by eigenvectors

- (c) **1 2 2** – compute 1 eigenvector that correspond to the first eigenvalue and 2 that correspond to the second
- (d) **1 2 3** – compute 1 eigenvector that correspond to the first, 1 that correspond to the second and 1 that correspond to the third eigenvalue
- (e) **1 1** – compute 2 eigenvectors that correspond to the first eigenvalue (2D picture)

### 4.3 Layers

Visible only if Draw partition is active. Draw in layers according to partition.

- **Type of Layout** – Select type of picture (2D – layers in y direction, or 3D – layers in z direction). According to that, appropriate menu appears.
- **In y direction** – Draw vertices in layers (y coordinate) inside layers draw vertices centered-equidistantly (x coordinate), z coordinate is 0.5 for all vertices.
- **In y direction+random in x** – Same as first option, only vertices are put on layers in random order not according to vertex numbers.

- **In z direction** – Draw layers in z direction, live x and y coordinates as they are.
- **In z direction + random in xy** – Draw layers in z direction, x and y coordinates get random values
- **Averaging x coordinate** – Use it after vertices are put on layers in 2D. Iteratively compute average x-coordinate of all neighbors and normalize. Good approximation of global picture, but vertices are put to close to each other. Use it on all vertices or only on selected one.
- **Averaging x and y coordinates** – Use it after vertices are put on layers in 3D. Iteratively compute average x and y coordinates of all neighbors and normalize. Good approximation of global picture, but vertices are put to close to each other. Use it on all vertices or only on selected one.
- **Tile in x direction** – After averaging x coordinate vertices are put to close to each other, so using this option vertices are repositioned to minimal distance described in resolution.
- **Tile in xy plane** – Same as previous, only this option is used when drawing 3D pictures.
- **Optimize layers in x direction** – Optimize layout in layers using minimization of the total length of lines.
  1. Forward – go from first to last layer. In the current layer optimize only layers having numbers equal or one smaller as the current layer number.
  2. Backward – go from last to first layer. In the current layer optimize only layers having numbers equal or one larger as the current layer number.
  3. Complete – go from first to last layer. In the current layer optimize only layers having numbers equal or one smaller or one larger as the current layer number.
- **Optimize layers in xy plane** – Same as previous, only this option is used when drawing 3D pictures.
- **Resolution** – How many additional positions are available on layers. Works only if Pgraph is selected. The higher the resolution, the better is the result of optimization, but also slower.

## 4.4 GraphOnly

Show complete picture without labels and arrows.

## 4.5 Previous

Draw previous network, and/or partition and/or vector which are loaded in **Pajek** (depending on selection in Options/PreviousNext/Apply to).

## 4.6 Redraw

Redraw network.

## 4.7 Next

Draw next network, and/or partition and/or vector which are loaded in **Pajek** (depending on selection in Options/ PreviousNext/ Apply to).

## 4.8 Options

Additional options for picture layout.

- **Transform** – Transformations of picture.
  1. **Fit area**
    - (a) **max(x), max(y), max(z)** – Draw picture as big as possible to fit area (resize each coordinate to fit in picture independently).
    - (b) **max(x,y,z)** – Resize to fit in area but keep real proportions (resize according to largest distance in all three coordinates, e.g. molecule).
  2. **Resize** – Resize picture (or selected part of it) in all three directions for selected factor.
  3. **Translate** – Translate picture (or selected part of it) in space.
  4. **Reflect y axis** – Reflect picture (or selected part of it) around y axis.
  5. **Rotate 2D** – Rotate picture (or selected part of it) in xy plane.
  6. **Fisheye** – Fisheye transformation (cartesian or polar) of picture. If no vertex is selected the middle point of picture (or selected part of it) is used as focus, otherwise first selected vertex will be used as focus.

- **Values of lines** – Meaning of values of lines during energy drawing or eigenvectors computing (no meaning, similarities, dissimilarities (distances)).
- **Mark vertices using** – Labels can be marked using
  1. labels
  2. numbers
  3. partition clusters (if partition of the same size is also selected)
  4. vector values (if vector of the same size is also selected)
  5. without labels
  6. without labels and arrows
  7. cluster only – only vertices belonging to the current Cluster are labeled in the layout.
- **Lines** – Select the way the lines are drawn:
  1. **Draw Lines**
    - (a) **Edges** – draw edges or not
    - (b) **Arcs** – draw arcs or not
    - (c) **Relations** – draw all lines (leave empty string) or just lines belonging to selected relations, e.g. 1-3,6,10-15.
  2. **Mark Lines**
    - (a) **No** – do not mark lines
    - (b) **with Labels**
    - (c) **with Values**
  3. **Different Widths** – if checked, the width of the lines will be determined by their line values.
  4. **GreyScale** – if checked, the color in grayscale of lines will be determined by their line values.
- **Size** – Determine size of vertices, width of vertices border, width of lines, size of arrows, size of font or turn them to default values.  
Sizes of vertices can be set to autosize (0-average). They can be read from input file (x\_fact and y\_fact) or determined by one vector (currently selected) or the two currently selected vectors. FontSize can be proportional to values stored in third partition.

- **Colors** – Determine color of background, vertices, border of vertices, edges, arcs and font (of vertices labels and lines labels) or turn them to default values. Colors of edges/arcs can represent relation number of lines. You can select which color should represent given class using **Relation Colors**. You can also use colors of vertices (**ic Red**), border of vertices (**bc Blue**), arcs and edges (**c Green**) as defined on input file (see Figure 19, page 83). FontColor can be determined by values stored in second partition.

Example NET file:

```
*Vertices          9
 1 "a" ic Pink      bc Black
 2 "b" ic Yellow    bc Cyan
 3 "c" ic Cyan      bc Yellow
 4 "d" ic Purple    bc Orange
 5 "e" ic Orange    bc Brown
 6 "f" ic Magenta   bc Green
 7 "g" ic Brown     bc Magenta
 8 "h" ic Red       bc Blue
 9 "i" ic Green     bc Magenta

*Arcs
 1      2      1 c Blue
 2      3      1 c Red
 3      4      1 c Black
 4      5      1 c Yellow
 5      6      1 c Gray
 6      7      1 c Cyan
 7      8      1 c Magenta
 8      9      1 c Purple
 9      1      1 c Brown
```

- **Layout** – Layout options
  1. **Redraw** – Redraw whole network *during* or/and *after* moving of selected vertex, and/or redraw if draw window is *paint*.
  2. **Real xy proportions** – The draw window has always square shape or not.
  3. **Arrows in the Middle** – Draw arrows in the middle of lines – not at terminal vertices.
  4. **Size of vertex 0** – How to handle vertices of size 0 (size of vertex is determined in input file or using vector)?
    - (a) **Hide vertex** – vertices of size 0 are shown or not.
    - (b) **Hide attached lines** – lines with one end in vertex of size 0 are shown or not.
  5. **Decimal Places** – How many decimal places to use when marking vertices using vectors.

6. **Show SubLabel** – Select the position of the vertices sublabel that is shown in network layouts.
- **ScrollBar On/Off** – Show/Hide the scrollbars in the top left corner of Draw window. When part of picture is selected, scrollbar is used for moving. When whole picture is selected, scrollbar is used for spinning (like pressing keys X, Y, Z, x, y, z – spinning around axis defined by the key, and S – spin around selected normal)
  - **Interrupt** – Interrupt period during optimisation (stop every ? second, or not)
  - **Previous/Next** – Select parameters when using Previous or Next commands for drawing sequence of networks in draw window:
    1. **Max. number** – How many networks to show in sequence. If the number is higher than number of existing networks the sequence will stop earlier.
    2. **Seconds to wait** – Seconds to wait between the two layouts.
    3. **Optimize Layouts** – Optimize the current layout or not. If the sequence of networks is obtained from the same network, it is useful to choose Energy/Starting Positions/ Given xy to start optimization with existing coordinates.
      - (a) **Kamada-Kawai** – Optimize the current layout using Kamada-Kawai algorithm.
      - (b) **2D Frucht. Rein.** – Optimize the current layout using 2D Frucht. Rein. algorithm.
      - (c) **3D Frucht. Rein.** – Optimize the current layout using 3D Frucht. Rein. algorithm.
      - (d) **No** – Do not optimize the layout, just show the picture.
    4. **Apply to** – Which object (Network, Partition, Vector) will change when Previous or Next is selected:
      - (a) **Network** – The previous / next network in memory is drawn. If Draw-Partition is selected and the new network matches in dimension with selected partition, the same partition will determine colors of vertices of the new network. If Draw-Vector is selected and the new network matches in dimension with selected vector, the same vector will determine sizes of vertices of the new network.  
Option can be used to show several networks of equal size using the same partition/vector.

- (b) **Partition** – The previous / next partition in memory is selected. If Draw-Partition is selected: The same network is drawn using previous / next partition (network and partition must match in size). Option can be used to show several partitions of selected network.
- (c) **Vector** – The previous / next vector in memory is selected. If Draw-Vector is selected: The same network is drawn using previous / next vector (network and vector must match in size). Option can be used to show several vectors of selected network.

By checking several objects (Network, Partition, Vector) at the same time previous / next networks will be drawn using previous / next partitions and (or) vectors at the same time. All consequent selected objects must match in size.

- **Select all** – Select all vertices in window (then possible to put vertices in given class).

## 4.9 Export

Export layout of the network to one of the following two or three dimensional formats:

- **2D** – two dimensional exports:
  - **EPS/PS** – Export to EPS format (with or without Clip, or WYSIWYG [What You See Is What You Get – exported EPS picture is similar to picture in Draw window – except that colors are Black/White or color determined by partition]). **PS** – Export to PS format (similar to EPS but without header).
  - **SVG** – Export to SVG (Scalable Vector Graphics) format. Additional controls over layout can be included in SVG or HTML. The plugin for examining layouts can be obtained from Adobe [1]. Linear or radial gradients (continuously smooth color transitions from one color to another) can be selected as well – up to three background colors can be selected in Export/Options window.
    1. **General** – Export to SVG without possibility to choose parts of the picture.
    2. **Labels/Arcs/Edges** – Export with possibility to turn labels, arcs and/or edges on/off.
    3. **Partition** – Export to SVG using one or two partitions. One partition is used by default: the same partition determines classes and

colors. But, if two partitions are defined by Partitions menu, first partition will determine classes, the second colors of vertices.

- (a) **Classes** – User can turn selected classes and lines among classes on/off.
  - (b) **Classes with semi-lines** – User can turn selected classes on/off. Lines among classes are drawn as semi-lines.
  - (c) **Nested Classes** – Upper classes are nested in lower – whenever selected class is turned on all higher classes are turned on too, and all lower classes are turned off (suitable for showing cores, for example).
4. **Line Values** – Export to SVG using values of lines. Threshold values or number of classes must be given. If you input #n, n classes of equal size will be generated. According to obtained thresholds, subsets of lines (and incident vertices) are defined and can be turned on/off inside web browser.
- (a) **Classes** – User can turn lines of selected value and incident vertices on/off.
  - (b) **Nested classes** – User can turn lines of selected value or higher (lower) and incident vertices on (off).
  - (c) **Options** – Additional options to emphasize the values of lines using some visual properties.
    - Different Colors** – Subsets of lines are drawn using colors which are used for partitioning vertices too.
    - Using GreyScale** – The darkness of a line corresponds to its value.
    - Different Widths** – The width of a line corresponds to its value.
5. **Multiple Relations Network** – Export multiple relations network to SVG. User can show/hide one or more relations in the layout.
6. **Current and all Subsequent** – If checked the current network and all subsequent networks will be exported to SVG. For each network separate html file is generated. Files are given the following names: file001.htm, file002.htm,... Generated html files get additional links (Previous/Next) to transition among them. If also next partitions / vectors fit in dimension to dimension of networks, partitions will determine color of vertices, vectors will determine sizes of vertices. *Subsequent* is applied to any combination of [Network, Partition, Vector] (one of the three objects only, any pair of them or all three of them) according to selection in *Options/Previous/Next/ Apply to* in Draw window.



- **Bitmap** – Export to Windows bitmap (bmp) format.
- **3D** – three dimensional exports:
  - **X3D** – Export to X3D (XML based 3D computer graphics, the successor of VRML) format.
  - **Kinemages** – Export to Kinemages format with balls or labels. You need Mage or King viewer to watch it. A free copy of the Mage software can be downloaded from its site [50].
    1. **Current Network Only** – Export only current network to Kinemages. Two partitions defined by Partitions menu can be used - one for generations one for colors.
    2. **Current and all Subsequent** – Export current network and all subsequent networks (use commands `KINEMAGE/Next` or `Ctrl N` in Mage). If also next partitions/vectors fit in dimension to dimension of networks, partitions will determine color of vertices, vectors will determine sizes of vertices. *Subsequent* is applied to any combination of [Network, Partition, Vector] (one of the three objects only, any pair of them or all three of them) according to selection in *Options/Previous/Next/ Apply to* in Draw window.
    3. **Multiple Relations Network** – Export with possibility to hide / show selected relations.
  - **VRML** – Export to VRML (Virtual Reality) format. For examining it you need a VRML viewer such as Cortona [23] or (older) Cosmo player [24].
  - **MDL MOL file** – Export to MDL Molfile format. You need Chime plugin (Chemscape Chime) for Netscape to explore it [43].
- **Options** – EPS, SVG, X3D and VRML default options (see section on **Exports to EPS/SVG/X3D/VRML**).
- **Append to Pajek project file** – Add current network to the end of selected project file (used by program **PajekToSvgAnim**).
  - **Select file** – Select project file.
  - **Append** – Append to project file.

### 4.10 Spin

- **Spin around** – Spin network around selected normal.
- **Perspective** – Distant vertices are drawn smaller (or not).
- **Normal** – Normal vector to spin around.
- **Step in degrees** – Step in degrees when showing rotation.

### 4.11 Move

Give additional constraints on hand vertex moving:

- **Fix** – Fix (do not allow) moving in x or y direction, or do not allow changing distance from center (circulating).
- **Grid** – Define  $(x, y)$  positions on grid, these become feasible positions for vertices during moving by hand.
- **Circle** – Define  $(x, y)$  positions on concentric circles, these become feasible positions for vertices during moving by hand.
- **Grasp** – Determine which additional vertices are moving when clicking with left mouse button close to vertex in given class. Vertices that will be moved are in the:
  1. **Closest Class Only**
  2. **Closest Class and Higher**
  3. **Closest Class and Lower**

### 4.12 Info

– Select aesthetic properties of the current layout to compute:

- Closest Vertices
- Smallest Angle
- Shortest/Longest Line
- Number of crossings if lines
- Vertex Closest to Line
- All Properties

Remember that coordinates of vertices must be between 0 and 1!

## 5 Exports to EPS/SVG/X3D/VRML

### 5.1 Defaults

If you have no labels in Draw window when you call Export there will also be no labels in EPS/SVG picture, otherwise numbers/labels like in Draw window will be shown. If you are looking at the picture using Draw/Partition, the same colors will be automatically used in EPS/SVG picture too.

### 5.2 Parameters in EPS, SVG, X3D, and VRML Defaults Window

Window is divided into 5 frames, two on the left and three on the right.

Note that settings you made in this window are overwritten if the parameters are specified in Pajek input (NET) file.

#### Top frame on the left – EPS/SVG Vertex Default

This frame defines default drawing of vertices when we export layouts to EPS and SVG:

- **Interior Color** – interior color of vertices (see Figure 19, page 83). If drawing using Partition is used, partition colors will overwrite the specified interior color.
- **Border Color** – color of the borderline of vertices.
- **Label Color** – color of label of vertices.
- **Border Width** – width of the borderline of vertices.
- **Label Position: Radius /Angle** – position where the label will be displayed:
  - **Radius** – distance of beginning of vertex label from vertex center – first polar parameter.
  - **Angle** – position of vertex label in degrees - second polar parameter (0..360).

If radius is equal to 0, the label will be centered, otherwise it will be left aligned on the specified position.

- **Fontsize** – size of font of vertices labels.

- **Label Angle** – angle in which vertex labels will be displayed: if angle is smaller than 360, it means relative according to horizontal line (0 – horizontally); otherwise it is relative to center of the layout (360 – concentrically). The latter is useful when all vertices are drawn in concentric circle(s) using **Layout/Circular**.
- **x/y Ratio** – ratio between size of vertex in  $x$  and  $y$  direction (e.g., value 1 for circle, value larger/smaller than 1 for ellipse).
- **Shape** – default shape of vertex (*ellipse, box, or diamond*).
- **Shapes file** – default shapes file, double click to change it.
- **Export options overwrite shapes file** – if checked: options selected in this window overwrite options defined in shapes file, otherwise default values for each shape are defined in selected shapes file.

#### **Bottom frame on the left – EPS/SVG Line Default**

This frame defines default drawing of lines when we export layouts to EPS and SVG:

- **Edge Color** – color of edges.
- **Edge Width** – width of edges.
- **Arc Color** – color of arcs.
- **Arc Width** – width of arcs.
- **Pattern** – pattern for drawing lines (*Solid or Dots*).
- **Arrow Size** – size of arrows.
- **Arrow Position** – distance of arrow from terminal vertex: If distance is between 0 and 1, it means relative distance according to arc length, e.g.: 0 – arrow touching the terminal vertex; 0.5 – arrow in the middle of the arc. If distance is larger than 1 it means absolute distance from the terminal vertex (this is useful if you want to have all arcs on the same distance from terminal vertex, regardless of arcs length)
- **Label Color** – color of line labels.
- **Label Angle** – angle in which label of line will be displayed: if angle is smaller than 360, it means relative to direction of line (0 – parallel to line); otherwise it is relative to horizontal line (360 – horizontally).

- **Label Position** – position of the center of line label – position is point on the line - distance of the center of line label from terminal vertex (see also **Arrow Position**)
- **Fontsize** – size of font of line labels.
- **Label Position: Radius /Angle** – position where the center of line label will be displayed according (relative) to **Label Position**:
  - **Radius** – distance of center of line label from point defined by **Label Position** - first polar parameter.
  - **Angle** – position of label in degrees - second polar parameter (0..360).
- **Only straight lines** – drawing bidirected and multiple lines without curves.

### Top frame on the right

This frame defines some additional defaults when we export layouts to EPS, SVG, X3D, or VRML:

- **EPS, SVG, X3D, VRML Size of Vertices** – default size of vertices when exporting to X3D/VRML (valid for EPS and SVG exports as well).
- **X3D/VRML Size of Lines** – width of lines in X3D/VRML.
- **EPS: Use RGB colors instead of CMYK** – in description of colors in EPS file RGB is used (default is CMYK).
- **SVG: 3D Effect on Vertices** – if selected, gradient will be applied to get 3D look of vertices.

### Middle frame on the right – *Background Colors*

This frame defines background color when exporting to EPS/SVG/X3D/VRML and gradients (continuously smooth color transitions from one color to another) when exporting layouts to SVG/X3D:

- **Bckg. Color 1** – Background color for layout in EPS/SVG/X3D/VRML.
- **Bckg. Color 2** – the second color for SVG/X3D export. *No* – means without second color, otherwise selected gradient will be used.
- **Bckg. Color 3** – the third color for SVG/X3D export – gradient.
- **Gradients** – type of gradient to use in SVG (*No*, *Linear*, or *Radial*). In X3D only *Radial* gradient is supported.

### Bottom frame on the right

This frame defines some additional defaults when we export layouts to EPS:

- **Left, Right, Top, Bottom** – additional border around layout when exporting to EPS (only when *EPS Clip* format is selected)
- **Border Color** – color of borderline of layout. It is used for export to SVG as well. *No* – means without borderline.
- **Border Radius** – radius of borderline of layout (if greater than 0 it means oval instead of rectangle).
- **Border Width** – width of borderline of layout.



Figure 17: *Spider / Križavec*; Photo: Stana.

### 5.3 Exporting pictures to EPS/SVG – defining parameters in input file

#### Definition of Network (and its picture) on Input ASCII File

For every vertex and line we can specify in details how it should be drawn (colors, shapes, sizes, patterns, rotations, widths...).

A kind of standardized language is used for describing networks. The following reserved words are used:

1. **\*Vertices**  $n$  – definition of vertices follows.  $n$  is number of vertices. Each vertex is described using following description line:

```
vertex_num label [x y z] [shape] [changes of default parameters]
```

Explanation:

- **vertex\_num** – vertex number (1, 2, 3...  $n$ )
  - **label** – if label starts with character A..Z or 0..9 first blank determines end of the label (e.g., vertex1), labels consisting of more words must be enclosed in pair of special characters (e.g., "vertex 1")
  - **x, y, z** – coordinates of vertex (between 0 and 1)
  - **shape** – shape of object which represents vertex. Shapes are defined in file SHAPES.CFG (ellipse, box, diamond, triangle, cross, empty)
- Description of parameters in shapes.cfg:

- **SHAPE**  $s$  –  $s$  is external name of vertex (used in Pajek network file)
- **sh** – sh can be ellipse, box, diamond, triangle, cross, empty. This is the name of PostScript procedure that actually draws object (procedure is defined in drawnet.pro).
- **s.size** – default size
- **x\_fact** – magnification in x direction
- **y\_fact** – magnification in y direction
- **phi** – rotation in degrees of object in + direction (0..360)
- **r** – parameter used for rectangle and diamond for describing radius of corners ( $r = 0$  – rectangle,  $r > 0$  – roundangle)
- **q** – parameter used for diamonds – ratio between top and middle side of diamond (try  $q = 0.01, q = 0.5, q = 2, \dots$ )
- **ic** – interior color of vertex. See Figure 19, page 83 for the list of possible colors.
- **bc** – boundary color of vertex

- **bw** – boundary width of vertex
- **lc** – label color
- **la** – label angle in degrees (0..360)
- **lr** – distance of beginning of vertex label from vertex center (radius – first polar parameter)
- **lphi** – position of label in degrees (0..360) (angle phi – second polar parameter)
- **fos** – font size
- **font** – PostScript font used for writing labels (Helvetica, Courier, ...)
- **HOOKS** – positions where edges can join the selected shape - according to *s\_size*. Three different ways to specify these positions:
  - (a) **CART** – *x y* – positions in Cartesian coordinates (*x,y*)
  - (b) **POLAR** – *r phi* – positions in polar coordinates, phi is positive angle (0..360)
  - (c) **CIRC** – *r phi1* – iteration of positions in polar coordinates *r* – radius,  $phi = k * phi1$ ,  $k = 1, 2, ..; k * phi1 \leq 360$

Default values can be changed for each vertex in definition line, example:

```
1 "vertex one" 0.3456 0.1234 0.5 box ic White fos 20
```

Explanation: White box will represent vertex 1, label (vertex one) will be displayed using font size 20.

2. **\*Arcs** (or **\*Edges**) – definition of arcs (edges). Format:

```
v1 v2 value [additional parameters]
```

Explanation:

- v1 – initial vertex number
- v2 – terminal vertex number
- value – value of arc from v1 to v2

These three parameters must always be present. If no other parameter is specified, the default arc will be black, straight, solid arc with following exceptions:

- if value is negative, dotted line will be used instead of solid,
- if arc is a loop (arc to itself) bezier loop will be drawn,
- if bidirected arc exists two curved bezier arcs will be drawn.



Arrow will be drawn at the end of the edge (at terminal vertex).

As we mentioned, hooks are used to specify exact position where line joins vertices.

Additional parameters:

- **w** – width of line
- **c** – color of line
- **p** – pattern of line (Solid, Dots)
- **s** – size of arrow
- **a** – type (shape) of arrow (A or B)
- **ap** – position of arrow
  - $ap = 0$  – arrow at terminal vertex
  - $0 < ap \leq 1$  – proportional distance from terminal vertex (according to line length)
  - $ap > 1$  – absolute distance
- **l** – line label (e.g. "line 1 2")
- **lp** – label position (look at  $ap$ )
- **lr** – label radius (position of center text from point on edge )
- **lphi** – label radius (angle of center text according to point on edge ) ( $lr$  and  $lphi$  are polar coordinates)
- **lc** – label color
- **la** – label angle
  - ( $0 < la < 360$  – relative to edge,  $la \geq 360$  – absolute angle according to x axis)
- **fos** – font size of label
- **font** – PostScript font used for writing labels (Helvetica, Courier, ...)
- **h1** – hook at initial vertex (0 – center, -1 the closest, 1, 2.. user defined)
- **h2** – hook at terminal vertex
- **a1** – angle at initial vertex (Bezier)
- **k1** – velocity at initial vertex (Bezier)
- **k2** – velocity at terminal vertex (Bezier)
- **a2** – angle at terminal vertex (Bezier)

Special shapes of lines can be defined using combinations of  $\alpha_1$ ,  $k_1$ ,  $\alpha_2$ ,  $k_2$ :

- $\alpha_1 = \alpha_2 = 0, k_1 \geq 0, k_2 \geq 0$  – straight line (default)
- $\alpha_1 = \alpha_2 = 0, k_1 = -1, k_2 > 0$  – oval edge with radius  $k_2$  (measure of radius is absolute as explained above)
- $\alpha_1 = \alpha_2 = 0, k_1 = -1, k_2 < 0$  – second possible oval edge with radius  $-k_2$
- $\alpha_1 = \alpha_2 = 0, k_1 = -2, k_2 > 0$  – circular arc with radius  $k_2$  in positive direction
- $\alpha_1 = \alpha_2 = 0, k_1 = -2, k_2 < 0$  – second possible circular arc with radius  $-k_2$  in positive direction
- $\alpha_1 = \alpha_2 = 0, k_1 = -3, k_2 > 0$  – circular arc with radius  $k_2$  in negative direction
- $\alpha_1 = \alpha_2 = 0, k_1 = -3, k_2 < 0$  – second possible circular arc with radius  $-k_2$  in negative direction
- $\alpha_1 = \alpha_2 = 0, k_1 = -4$  – double edge
- $\alpha_1$  or  $\alpha_2 \neq 0, k_1 > 0, k_2 > 0$  – Bezier curve (if  $\alpha_1$  and  $\alpha_2$  have different signs line goes from one to another side of straight line connecting both vertices, if  $\alpha_1$  and  $\alpha_2$  have the same sign – line stays on the same side of straight line connecting both vertices)

### 3. \*Edges – definition of edges.

The same parameters as for arcs can be used, except that type ( $a$ ), size ( $s$ ) and position ( $ap$ ) have no meaning for edges.

## PS and EPS

- PS – Export to Postscript (.PS) without header (drawnet.pro). Use it to spare space, if you have many pictures and your word processor enables you to define the header separately (like in L<sup>A</sup>T<sub>E</sub>X).
- EPS – Export into file with Encapsulated PostScript description (.EPS). Drawnet.pro is already inserted in the beginning. The picture is complete, you can include it into text, make it bigger or smaller (without losing quality), rotate it, print it on Postscript printer, view it with GhostScript viewer, convert it to PDF, JPG...

## An example

```
*Vertices 4
1 "ellipse" 0.120 0.285 0.5 ellipse_x_fact 5 y_fact 3 fos 20 ic LightYellow lc Red
2 "box" 0.818 0.246 0.5 box_x_fact 5 y_fact 3 fos 20 ic LightCyan lc Blue
3 "diamond" 0.368 0.779 0.5 diamond_x_fact 7 y_fact 3 fos 20 ic LightGreen lc Black
4 "triangle" 0.835 0.833 0.5 triangle_x_fact 9 y_fact 3 fos 20 ic LightOrange lc Brown lr 30 lphi 200
*Arcs
1 1 1 h2 0 w 3 c Blue s 2 a1 -130 k1 0.6 a2 -130 k2 0.6 ap 0.25 l "Bezier loop" lc OliveGreen fos 20 lr 13 lp 0.5 la 360
2 1 1 h2 1 a1 120 k1 1 a2 10 k2 0.8 ap 0 1 "Bezier arc" lphi 270 la 180 lr 13 lp 0.5
1 2 1 h2 -1 a1 40 k1 2 a2 -30 k2 0.8 ap 0 1 "Bezier arc" lphi 90 la 0 lp 0.75
4 2 -1 1 "Straight dotted arc" p Dots c Red
*Edges
1 3 1 1 "Straight edge" lp 0.4
3 4 1 1 "Straight edge"
```

You have to set some options in Pajek's draw window: for example **Options / Lines / Mark Lines / with labels** to activate the display of line labels.

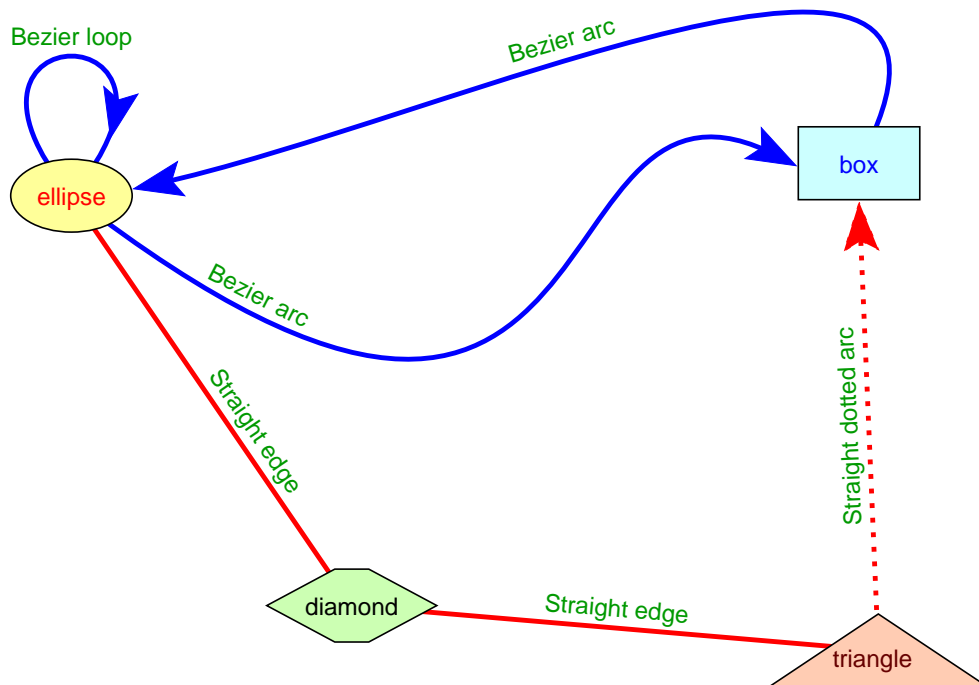


Figure 18: Example picture

## 6 Using Macros in **Pajek**

### 6.1 What is a Macro?

A macro enables you to record a sequence of primitive **Pajek** commands into a file. You can use this file later to execute the saved sequence of commands without selecting one by one.

### 6.2 How to record a Macro?

1. First load all objects (networks, partitions,...) which will be used by a macro.
2. Select objects which will be used in macro (for example, network which will be first used must be shown in Network ComboBox).
3. Choose Macro/Record and select the name of macro file (default extension is `.mcr`).
4. Use **Pajek** as usual to define a sequence of commands. We advise to add additional comments to macro file using Macro/Add Message. When running macro this comments are displayed in Report window and help you to follow the macro execution.
5. At the end select Macro/Record again to stop recording.

### 6.3 How to execute the Macro?

1. First load object(s) that will be used as input in macro execution.
2. Choose Macro/Play and select the macro file.

### 6.4 Example

The following macro performs topological sort of an acyclic network:

```
NETBEGIN 2
CLUBEGIN 1
PERBEGIN 1
CLSBEGIN 1
HIEBEGIN 1
VECBEGIN 1
NETPARAM 1
```

```

Msg Depth Partition
C 1 DEP 1 (10)
Msg Make Permutation
P 1 MPER 1 (10)
Msg Reordering network
N 2 REOR 1 1 (10)

```

The first seven commands store the current state of ComboBoxes. The acyclic network on which we want to execute topological sort must be at the top of Network ComboBox before starting the macro.

## 6.5 List of macros available in installation file

### 6.5.1 Macros prepared for genealogies and other acyclic networks

- **Path** – find shortest chain (undirected path) between the two vertices.
- **Descendants** – find all vertices 'later' than selected vertex (in a p-graph).
- **Ancestors** – find all vertices 'before' the selected vertex (in a p-graph).
- **Cognatic3** – find vertices three generations 'before' and three generations 'after' the selected vertex (in a p-graph).
- **Cognatic** – find all reachable vertices 'before' and all 'after' the selected vertex (in a p-graph).
- **Layers, Layers1, Layers2** – draw acyclic network in layers (different algorithms).
- **zLayers** – draw acyclic network in layers in  $z$  direction.
- **LongestPatrilineage** – find the longest patrilineage in Ore-genealogy. The genealogy must be read with the option **Ore graph: 1-Male, 2-Female links checked**.
- **LongestMatrilineage** – find the longest matrilineage in Ore-genealogy. The genealogy must be read with the option **Ore graph: 1-Male, 2-Female links checked**.
- **NumDescendants** – for each vertex compute the size of output domain (number of descendants in Ore-graph).
- **NumAncestors** – for each vertex compute the size of input domain (number of ancestors in Ore-graph).

### 6.5.2 Macros prepared for computing derived kinship relations

Pajek generates three relations when reading genealogy as Ore graph with the option **Ore graph: 1-Male, 2-Female links checked:**

- 1 : *is a father of*
- 2 : *is a mother of*
- 3 : *is a spouse of*

Other kinship relations can be obtained from these relations by running macros. Gender partition is also result of reading genealogy and is input to the following macros.

- 4 : *is a parent of*
- 5 : *is a child of*
- 6 : *is a son of*
- 7 : *is a daughter of*
- 8 : *is a husband of*
- 9 : *is a wife of*
- 10 : *is a sibling of*
- 11 : *is a brother of*
- 12 : *is a sister of*
- 13 : *is an uncle of*
- 14 : *is an aunt of*
- 15 : *is a semisibling of*

Using macro **add\_all\_relations** you can add all above relations at once.

As a test/example file you can use **\*/pajek/data/family.ged**

## 6.6 Repeating last command

Macro submenu enables to run the last command executed by Pajek several times applied to different successive objects, as well.

Example: after loading 100 networks in Pajek, execute degree partition on the first network and run Repeat Last Command by typing 99 to compute degree partition on all other networks.

Commands that include several objects can be run as well (e.g. extracting sub-networks according to selected partition(s)). There are 3 different possibilities for extracting from the set of networks (possibilities are selected by fixing appropriate objects):

- for all networks extracting is determined by the same partition (incrementing applied to network, not applied to partition – partition is fixed)

- for all networks extracting is determined by another partition (incrementing applied to network and to partition – nothing is fixed)
- for one network several extractions are determined by different partitions (incrementing not applied to network, applied to partition – network is fixed)

*Important:* Always first execute the command on the first loaded object(s). Repeating will start with object(s) that have object number(s) one higher than the object(s) on which the command was executed.

If the result of the command is also a constant, all constants are stored in a vector. In Pajek the following constants exist: number of vertices, arcs, edges, network densities, centralization indices, diameter, relinking index, correlation and contingency coefficients, number of fragments, main core number, number of components, length of critical path, maximum flow, distance between vertices, number of islands, minimum and maximum value in partition / hierarchy, minimum, maximum, arithmetic mean, median and standard deviation in vector.

## 7 Blockmodeling in Pajek

The blockmodeling option is an embedding of the programs for optimization approach to generalized blockmodeling `MODEL2` and `TWOMODEL` from package `STRAN – STRucture ANalysis` [8] into **Pajek**.

The blockmodeling command seeks for the best partition of a given network satisfying given types of blocks – generalized blockmodeling [6, 29]. The blockmodel can be built inside Pajek (if *User Defined* is selected) and/or its description can be stored in MDL file. The impact of errors in each block can be controlled using penalty weights.

The option supports also generalized blockmodeling of two-mode networks [28].

The maximum size of a network on which the command can be applied is 250 vertices. But the real limit is time complexity – already on 100 vertices optimization can last some hours.

The results of the command are stored as partitions. They can be displayed as a picture

```
Draw / Draw-Partition
and
Layout / Energy / Kamada-Kawai / Free
or
Layout / Circular / using Partition
```

The result can be displayed also in the matrix form. This requires two steps:

```
Partition / Make Permutation
File / Network / Export Matrix to EPS /
Using Permutation enter file name; yes
```

### 7.1 MDL files

The structure of a MDL file is evident from the following example

```
*MODEL Tina
9
0 3 100 0 1 2 3 4
*CONSTRAINTS
1 100 2 1
4 100 1 3
*EOM
```

**The first character in each line should be a star \* or a blank.  
The last character in a line should not be a blank.**



A number in the second line is number of clusters. In the case of **two-mode network** two numbers should be given in this line – number of row-clusters and number of col-clusters. The other data are the same for both type of networks. The following lines have the structure

$$i \quad j \quad \text{penalty} \quad t_1 \quad t_2 \dots t_k$$

When  $i, j > 0$  the line prescribes that the block  $(i, j)$  can be of types  $t_1, t_2 \dots t_k$ . The types are coded as follows

0	-	-	null	7	rfn	-	row-function
1	com	-	complete	8	cfm	-	col-function
2	rdo	-	row-dominant	9	den	-	density
3	cdo	-	col-dominant	10	dnc	-	do not care
4	reg	-	regular	11	one	-	non-null
5	rre	-	row-regular	12	sym	-	symmetric
6	cre	-	col-regular				

Lines with  $i = 0$  defines the types of parts of model matrix:

- $j = 0$ : diagonal (for one-mode networks only);
- $j = 1$ : upper triangle (for one-mode networks only);
- $j = 2$ : lower triangle (for one-mode networks only);
- $j = 3$ : complete matrix.

**In the case of several lines describing the same block the last prescription prevails.**

Lines following \*CONSTRAINTS define additional constraints in blockmodel. Constraints have the form

$$k \quad \text{penalty} \quad i \quad j$$

with the following meaning

- $k = 1$ :  $i \in C_j$  – vertex  $i$  belongs to cluster  $C_j$ ;
- $k = 2$ :  $i \notin C_j$  – vertex  $i$  does not belong to cluster  $C_j$ ;
- $k = 3$ :  $C(i) = C(j)$  – vertices  $i$  and  $j$  belong to the same cluster;
- $k = 4$ :  $C(i) \neq C(j)$  – vertices  $i$  and  $j$  belong to different clusters;
- $k = 5$ :  $i \leq |C(j)|$  – cluster  $C_j$  contains at least  $i$  vertices;

- $k = 6$ :  $i \geq |C(j)|$  – cluster  $C_j$  contains at most  $i$  vertices.

Be careful in the case of two mode network for constraints of type  $k = 5$  and  $k = 6$  (constraints on min/max number of vertices in selected cluster): If there are  $r$  row-clusters and  $c$  col-clusters in blockmodel, then use numbers in the range  $1 \dots r$  to define constraint on row-cluster and numbers in the range  $r + 1 \dots r + c$  to define constraint on col-cluster.

The violations of constraints contribute to criterion function with a term

$$+ \quad \# \text{ of violations} \quad \times \quad \text{penalty}$$

The values of penalties have to be in the range 0 to 1000.

## 7.2 Examples of MDL files

### 7.2.1 Regular blocks

```
*MODEL Regular
10
0 3 1 0 1 4
*EOM
```

### 7.2.2 Diagonal blocks (clustering)

```
*MODEL Diagonal
10
0 3 100 0
0 0 1 0 1 4
*EOM
```

### 7.2.3 Acyclic model (up)

```
*MODEL Hierarchy
9
0 1 1 0 5 6
0 0 10 0 1 4 12
0 2 100 0
*EOM
```

### 7.2.4 Acyclic model with symmetric clusters (down)

```
*MODEL SymHiera
9
```

```
0 0 10 0 1 12
0 1 100 0
0 2 1 0 11
*EOM
```

### 7.2.5 Center-Periphery

```
*MODEL Center-Periphery
2
0 3 1 0 11
2 2 10 0
1 1 100 0 1 4
*EOM
```

### 7.2.6 Regular path

```
*MODEL Regular Path
9
0 0 10 0 1 4
1 2 10 0 1 4
2 3 10 0 1 4
3 4 10 0 1 4
4 5 10 0 1 4
5 6 10 0 1 4
6 7 10 0 1 4
7 8 10 0 1 4
8 9 10 0 1 4
*EOM
```

### 7.2.7 Regular chain

```
*MODEL Regular Chain
9
0 0 10 0 1 4
1 2 10 0 1 4
2 3 10 0 1 4
3 4 10 0 1 4
4 5 10 0 1 4
5 6 10 0 1 4
6 7 10 0 1 4
7 8 10 0 1 4
```

```
8 9 10 0 1 4
2 1 10 0 1 4
3 2 10 0 1 4
4 3 10 0 1 4
5 4 10 0 1 4
6 5 10 0 1 4
7 6 10 0 1 4
8 7 10 0 1 4
9 8 10 0 1 4
```

\*EOM

### 7.2.8 2-mode 'standard model' for Davis.net

```
*MODEL UserDefined
2 3
  1 1          1 1
  1 2          1 1
  1 3         100 0
  2 1         100 0
  2 2          1 1
  2 3          1 1
```

\*EOM

## 8 Colors in Pajek

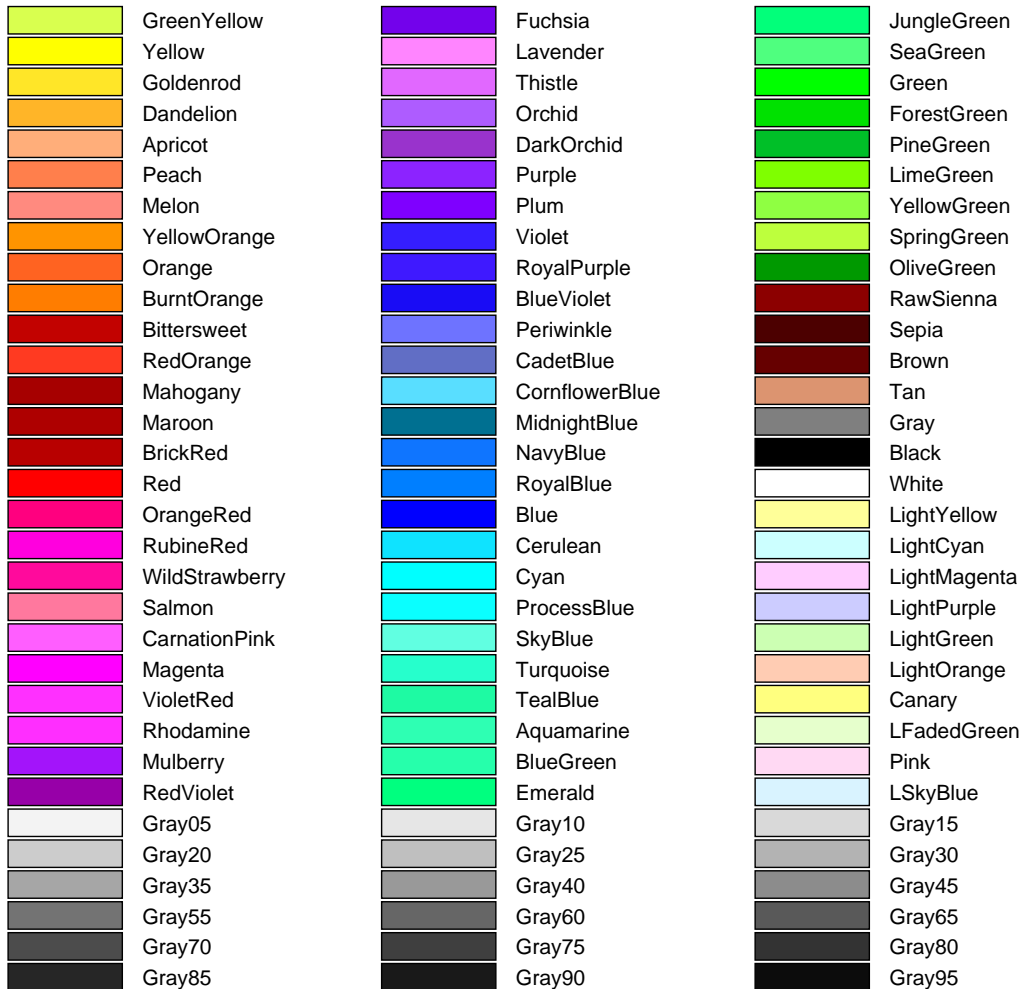


Figure 19: Colors in Pajek.

	0 - Cyan		20 - WildStrawberry
	1 - Yellow		21 - ForestGreen
	2 - LimeGreen		22 - Salmon
	3 - Red		23 - LSkyBlue
	4 - Blue		24 - GreenYellow
	5 - Pink		25 - Lavender
	6 - White		26 - LFadedGreen
	7 - Orange		27 - LightPurple
	8 - Purple		28 - CornflowerBlue
	9 - CadetBlue		29 - LightOrange
	10 - TealBlue		30 - Tan
	11 - OliveGreen		31 - LightCyan
	12 - Gray		32 - Gray20
	13 - Black		33 - Gray60
	14 - Maroon		34 - Gray40
	15 - LightGreen		35 - Gray75
	16 - LightYellow		36 - Gray10
	17 - Magenta		37 - Gray85
	18 - MidnightBlue		38 - Gray30
	19 - Dandelion		39 - Gray70

Figure 20: Partition colors.

## 9 Citing Pajek

For citing **Pajek** you may consider:

- V. Batagelj, A. Mrvar: *Pajek – Program for Large Network Analysis*. Home page: <http://vlado.fmf.uni-lj.si/pub/networks/pajek/>
- W. de Nooy, A. Mrvar, V. Batagelj: *Exploratory Social Network Analysis with Pajek*, Structural Analysis in the Social Sciences 27, Cambridge University Press, 2005. ISBN:0521602629. **CUP**, **Amazon**.
- V. Batagelj, A. Mrvar: *Pajek – Analysis and Visualization of Large Networks*. In Jünger, M., Mutzel, P. (Eds.): *Graph Drawing Software*. Springer (series Mathematics and Visualization), Berlin 2003. 77-103. ISBN 3-540-00881-0. **Springer**, **Amazon**, **preprint**
- V. Batagelj, A. Mrvar: *Pajek – Program for Large Network Analysis*. *Connections*, **21**(1998)2, 47-57. **preprint**



Figure 21: *Gartenkreuzspinne / Araneus diadematus*; **Photo:** Stefan Ernst

## References

- [1] Adobe SVG viewer. <http://www.adobe.com/svg/>  
<http://www.adobe.com/svg/viewer/install>
- [2] Albert R., Barabasi A.L.: Topology of evolving networks: local events and universality.  
<http://xxx.lanl.gov/abs/cond-mat/0005085>
- [3] Batagelj V.: Papers on network analysis.  
<http://vlado.fmf.uni-lj.si/pub/networks/doc/>
- [4] Batagelj V.: Workshop on Network Analysis, Sydney, Australia: 14th to 17th June 2005; at Nicta (National ICT Australia).  
<http://vlado.fmf.uni-lj.si/pub/networks/doc/#NICTA>
- [5] Batagelj V.: Some new procedures in Pajek. Dagstuhl seminar 05361, Dagstuhl, Germany, Sept 5-9, 2005.  
<http://vlado.fmf.uni-lj.si/pub/networks/doc/dagstuhl/NewProcs.pdf>
- [6] Batagelj, V. (1997) Notes on blockmodeling. *Social Networks* **19**, 143-155.
- [7] Batagelj V.: Efficient Algorithms for Citation Network Analysis.  
<http://arxiv.org/abs/cs.DL/0309023>
- [8] Batagelj V.: **MODEL 2**. <http://vlado.fmf.uni-lj.si/pub/networks/>
- [9] Batagelj V., Brandes U. (2005): Efficient Generation of Large Random Networks. *Physical Review E* **71**, 036113, 1-5.
- [10] Batagelj, V., Ferligoj, A., and Doreian, P. (1992), Direct and Indirect Methods for Structural Equivalence, *Social Networks*, **14**, 63–90.
- [11] Batagelj, V., Doreian, P., and Ferligoj, A. (1992) An Optimizational Approach to Regular Equivalence. *Social Networks* **14**, 121-135.
- [12] Batagelj V., Mrvar A.: Pajek.  
<http://vlado.fmf.uni-lj.si/pub/networks/pajek/>
- [13] Batagelj V., Mrvar A. (2000) Some Analyses of Erdős Collaboration Graph. *Social Networks*, **22**, 173-186
- [14] Batagelj V., Mrvar A. (2001) A Subquadratic Triad Census Algorithm for Large Sparse Networks with Small Maximum Degree. *Social Networks*, **23**, 237-243



- [15] Batagelj V., Mrvar A., Zaveršnik M. (1999) Partitioning Approach to Visualization of Large Graphs. In: Kratochvíl J. (Ed.) GD'99, Štířín Castle, Czech Republic. LNCS 1731. Springer-Verlag, 90-97.
- [16] Batagelj V., Mrvar A., Zaveršnik M. (2002) Network analysis of texts. Language Technologies, Ljubljana, p. 143-148. <http://nl.ijs.si/isjt02/zbornik/sdjt02-24bbatagelj.pdf>
- [17] Batagelj V., Zaveršnik M. (2002): Generalized Cores. <http://arxiv.org/abs/cs.DS/0202039>
- [18] Batagelj, V. and Zaveršnik, M. (2003): Short Cycles Connectivity. <http://arxiv.org/abs/cs.DS/0308011>
- [19] Bollobas B.: Random Walks on Graphs,
- [20] Broder A. et al. (2000): Graph structure in the web. <http://www.almaden.ibm.com/cs/k53/www9.final/>
- [21] Burt R.S. (1992): *Structural Holes*. The Social Structure of Competition. Cambridge MA: Harvard University Press.
- [22] Butts, C.T. (2002) sna: Tools for Social Network Analysis. <http://cran.at.r-project.org/src/contrib/PACKAGES.html#sna>
- [23] Cortona Player (2006) <http://www.parallelgraphics.com/products/cortona/>
- [24] Cosmo Player (2002) <http://ca.com/cosmo/>
- [25] de Nooy W., Mrvar A., Batagelj V. (2002) *Exploratory Social Network Analysis With Pajek*. Structural Analysis in the Social Sciences 27, Cambridge University Press, 2005. ISBN:0521602629. CUP, Amazon.
- [26] Doreian P., Mrvar A. (1996) A Partitioning Approach to Structural Balance. *Social Networks*, 18. 149-168
- [27] Doreian, P., Batagelj, V., Ferligoj, A. (2000) Symmetric-acyclic decompositions of networks. *J. classif.*, 17(1), 3-28.
- [28] Doreian P., Batagelj V., Ferligoj A. (2004) Generalized blockmodeling of two-mode network data. *Social Networks* 26, 29-53.
- [29] Doreian P., Batagelj V., Ferligoj A.: *Generalized Blockmodeling*, Structural Analysis in the Social Sciences 25, Cambridge University Press, 2005. ISBN:0521840856. CUP, Amazon.

- [30] Dremelj P., Mrvar A., Batagelj V. (2002) Analiza rodoslova dubrovačkog vlasteoskog kruga pomoću programa **Pajek**. Anali Dubrovnik XL, HAZU, Zagreb, Dubrovnik, 105-126 (in Croat).
- [31] GEDCOM 5.5.  
<http://homepages.rootsweb.com/~pmcbride/gedcom/55gctoc.htm>
- [32] Ghostscript, Ghostview and GSview. <http://www.cs.wisc.edu/~ghost/>
- [33] Gibbons A. (1985) Algorithmic Graph Theory. Cambridge University Press.
- [34] Grossman J. (2002) The Erdős Number Project.  
<http://www.oakland.edu/~grossman/erdoshp.html>
- [35] Hall, B.H., Jaffe, A.B. and Tratjenberg M.: The NBER U.S. Patent Citations Data File. NBER Working Paper 8498 (2001).  
<http://www.nber.org/patents/>
- [36] Hamberger, K., Houseman, M., Daillant, I., White, D.R., Barry, L. (2004) *Matrimonial Ring Structures*. Math. & Sci. hum. / Mathematics and Social Sciences, **42(4)**, 83-119.  
<http://www.ehess.fr/revue-msh/pdf/N168R965.pdf>
- [37] Hummon, N.P., Doreian, P. (1989) Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**, 39–63.
- [38] Jones B. (2002). Computational geometry database.  
<http://compgeom.cs.uiuc.edu/~jeffe/compgeom/biblios.html>
- [39] Kleinberg J. (1998) Authoritative sources in a hyperlinked environment. In Proc 9th ACM/SIAM Symposium on Discrete Algorithms, p. 668-677.  
<http://www.cs.cornell.edu/home/kleinber/auth.ps>
- [40] Knuth, D. E. (1993) *The Stanford GraphBase*. Stanford University, ACM Press, New York. <ftp://labrea.stanford.edu/pub/sgb/>
- [41] LOCKS: CRA Analyses of News Stories on the Terrorist Attack. Arizona State University. <http://locks.asu.edu/terror/>
- [42] McCabe, T. Computer Science Approaches: Visualization Tools and Software Metrics. in Survey Automation. NAP, 2003, p. 116-136.  
<http://books.nap.edu/books/0309089301/html/116.html>
- [43] MDL Information Systems, Inc. (2002) <http://www.mdli.com/>

- [44] James Moody home page (2002)  
<http://www.soc.sbs.ohio-state.edu/jwm/>
- [45] Murtagh, F. (1985) Multidimensional Clustering Algorithms, *Compstat lectures*, **4**, Vienna: Physica-Verlag.
- [46] Pajek's datasets:  
<http://vlado.fmf.uni-lj.si/pub/networks/data/>
- [47] D.M. Pennock et al. (2002) Winners don't take all, *PNAS*, **99**/8, 5207-5211.
- [48] Peterson J. L.: *Petri Net Theory and the Modeling of Systems*.
- [49] The R Project for Statistical Computing. <http://www.r-project.org/>
- [50] Richardson D.C., Richardson J.S. (2002) The Mage Page.  
<http://kinemage.biochem.duke.edu/index.html>
- [51] Scott, J. (2000) *Social Network Analysis: A Handbook*, 2nd edition. London: Sage Publications.
- [52] Seidman S. B. (1983) Network structure and minimum degree, *Social Networks*, **5**, 269–287.
- [53] Tarjan, R. E. (1983) *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics Philadelphia, Pennsylvania.
- [54] UCINET (2002) <http://www.analytictech.com/>
- [55] The United States Patent and Trademark Office.  
<http://patft.uspto.gov/netahtml/srchnum.htm>
- [56] Wasserman S., Faust K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge.
- [57] White D.R., Batagelj V., Mrvar A. (1999) Analyzing Large Kinship and Marriage Networks with Pgraph and **Pajek**. *Social Science Computer Review*, **17** (3), 245-274
- [58] Wilson, R.J., Watkins, J.J. (1990) *Graphs: An Introductory Approach*. New York: John Wiley and Sons.
- [59] W3C SVG page. <http://www.w3.org/Graphics/SVG>