Pajek

Program for Analysis and Visualization of Large Networks

Reference Manual

List of commands with short explanation

version BE

Vladimir Batagelj and Andrej Mrvar

Ljubljana, September 28, 2004

PdfLaTex version October 1, 2003

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

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

vladimir.batagelj@uni-lj.si

Andrej Mrvar
Faculty of Social Sciences
University of Ljubljana, Slovenia

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

andrej.mrvar@uni-lj.si

Contents

1 Data objects

Pajek is a program for analysing large networks (networks having some hundred of thousands of vertices).

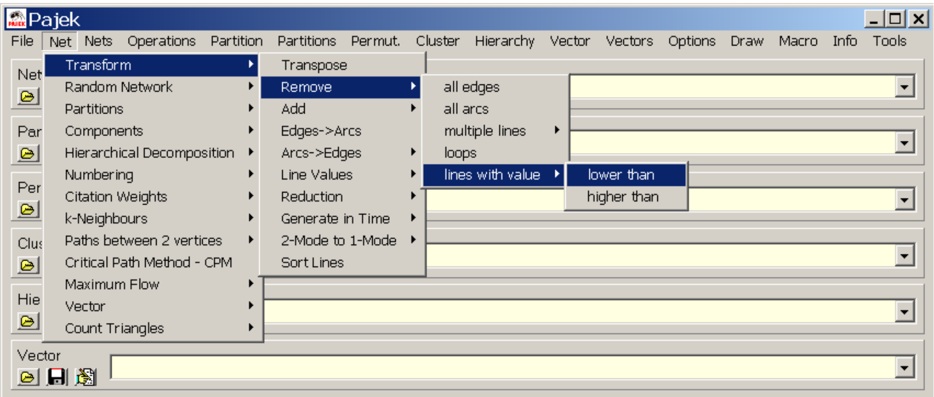


Figure 1: **Pajek**'s Main Window

Six objects are used:

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 arcslists/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 1) 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.

2 Main Window Tools

2.1 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. If you want to draw lines among classes of 2-mode network, you must define affiliation partition as first and partition containing classes as the second partition (in **partitions** menu).
    - **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).
    - **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*

  – **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.
  - **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.
  - **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.
  - **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).
  - **Save** selected hierarchy to Ascii file.
  - **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.
  - **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.

2.2 Net

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

- **Transform**
  - **Transpose** – Inverse (transpose) network of selected network (change direction of arrows in directed network).
  - **Remove**
    * **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.
    * **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.
  - **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).
    * **Sibling edges** – Add sibling edges to vertices with a common
      1. **Input** – arc-ancestor
      2. **Output** – arc-descendant

    * **Vertices 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.
- **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.
- **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**
    * **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).

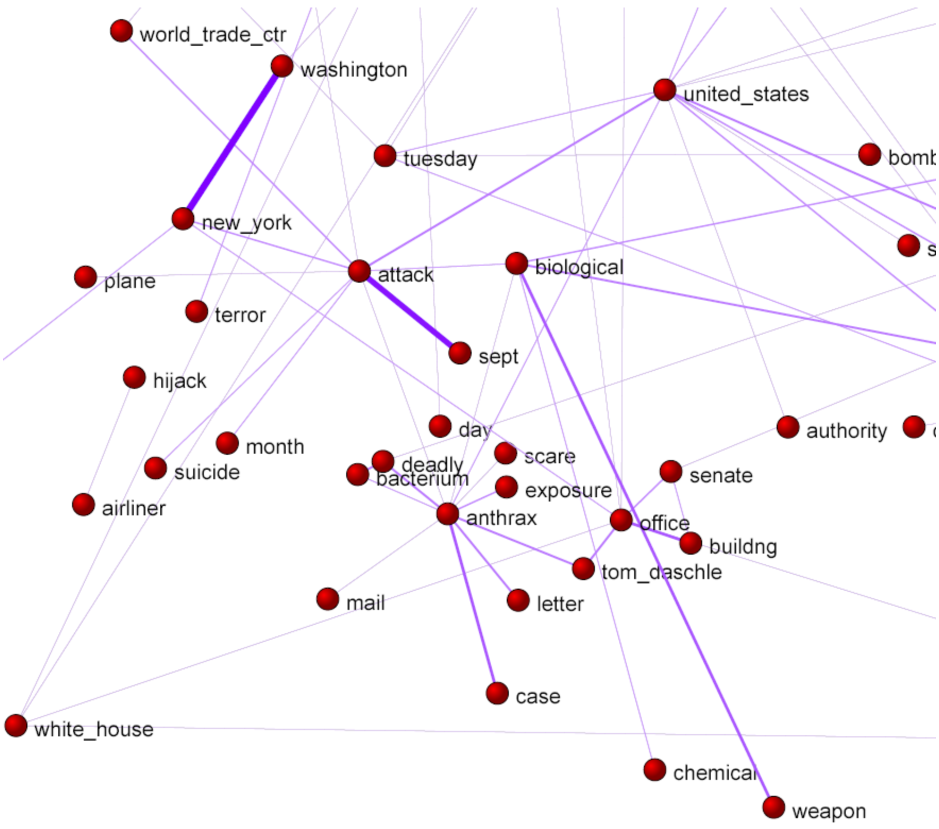


Figure 2: Part of Reuters Terror News network on the 36th day.

* ∗ **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. Result is simpler network (for drawing). Original network cannot be restored!
* ∗ **Degree** – (Recursively) delete from network all vertices with degree lower than selected value (according to Input, Output or All degree). Original network cannot be restored!
* ∗ **Design (flow graph)** Reduction of all structural parts of network according to McCabe (for programs – flow graphs) [35].

– **Generate in Time** – Generate network in specified time(s). 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.

– **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.
* **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:

$$
\begin{aligned}
\text{Geo}_{ij} &= \frac{a_{ij}}{\sqrt{a_{ii}a_{jj}}} \\
\text{Input}_{ij} &= \frac{a_{ij}}{a_{jj}} \\
\text{Output}_{ij} &= \frac{a_{ij}}{a_{ii}} \\
\text{Min}_{ij} &= \frac{a_{ij}}{\min(a_{ii}, a_{jj})} \\
\text{Max}_{ij} &= \frac{a_{ij}}{\max(a_{ii}, a_{jj})} \\
\text{MinDir}_{ij} &= \begin{cases} \frac{a_{ij}}{a_{ii}} & a_{ii} \leq a_{jj} \\ 0 & \text{otherwise} \end{cases} \\
\text{MaxDir}_{ij} &= \begin{cases} \frac{a_{ij}}{a_{jj}} & a_{ii} \leq a_{jj} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

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

– **Sort Lines** – For each vertex sort lines connected to it in increasing order according to vertex on the other side of line.

- **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 **Pajek** a more intuitive *average degree* $\overline{d}$ is used. They are connected with relations $\overline{d} = \frac{1}{n}\sum_{v\in V}\deg(v) = \frac{2m}{n}$ 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 [40]. 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

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

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

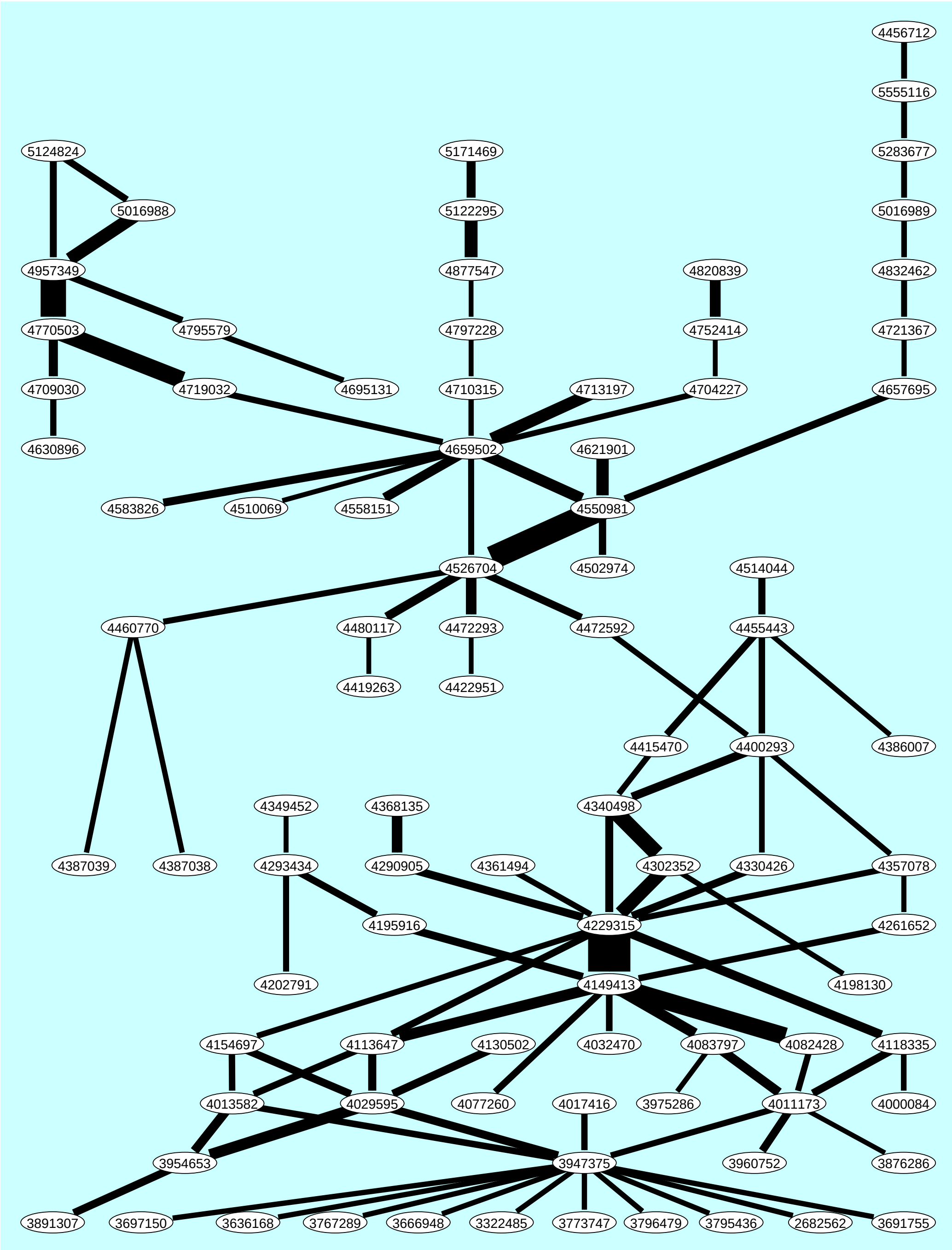


Figure 3: US Patents – Main island 'liquid-crystal display'

  * **All** ... all neighbors.

- **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.

  Additionally, 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).
- **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]**

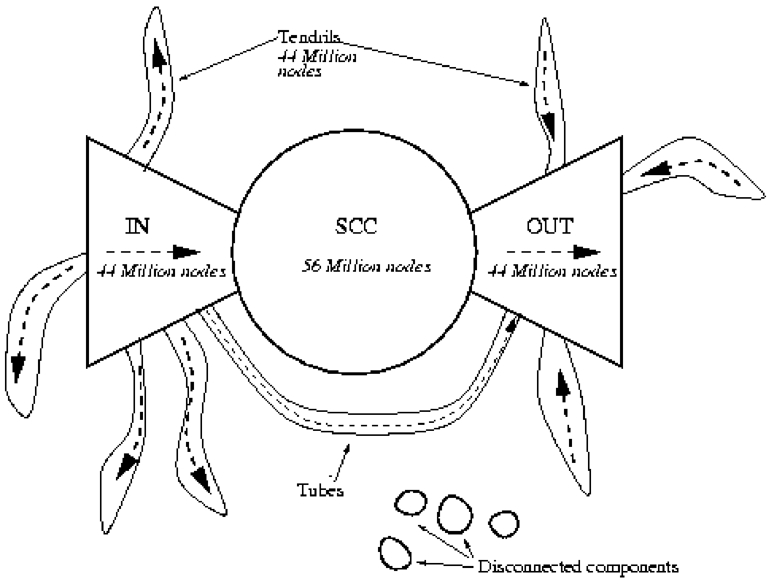


Figure 4: Bow-tie – Graph structure in the web [16]

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

- **Bow-Tie** – Partition vertices of directed network to 'peripheral' (1 – input acyclic vertices, 2 – output acyclic vertices, 3 – acyclic 'tube' vertices) and 4 – 'central' vertices.

- **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* 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 [22].
- **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).
  - **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 [30]:
  - **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 [49, 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* [32]. 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) [17, 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.

$$c_{ij} = (c_{ij} + \sum_k p_{ik} p_{kj})^2$$

    * vector containing aggregate constraint C_i: $C_i = \sum_j c_{ij}$

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

  Let $\deg(v)$ denotes degree of vertex v, $|E(G_1(v))|$ number of lines among vertices in 1-neighborhood of vertex v, 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:

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

  * CC_2 – coefficients considering 2-neighborhood

  $$CC_2(v) = \frac{|E(G_1(v))|}{|E(G_2(v))|} \qquad CC_2'(v) = \frac{\deg(v)}{\mathrm{MaxDeg}} CC_2(v)$$

  If $\deg(v) \leq 1$ 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 Triangles** – For each line count number of triangles to which the line belongs.

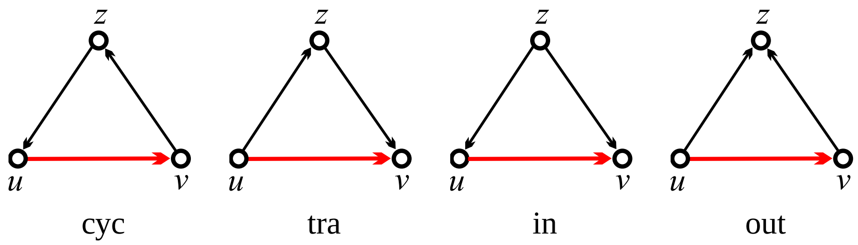


Figure 5: Types of directed triangles on an arc

- **Undirected** – for undirected networks – count undirected triangles.
- **Directed** – for directed networks – count **cyc**lic, **tra**nsitive, **in**put and **out**put or all triangles (see Figure 5).

2.3 Nets

Binary operations on networks. Two networks must be selected before performing binary operations.

- **First Network** – Network that is currently active, will be used as first network in operations.
- **Second Network** – Network that is currently active, will be used as second network in operations.
- **Union of lines** – Fuse selected networks. 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.
- **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.
- **Difference** – Difference of selected networks.
- **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 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.
      · **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

- **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**

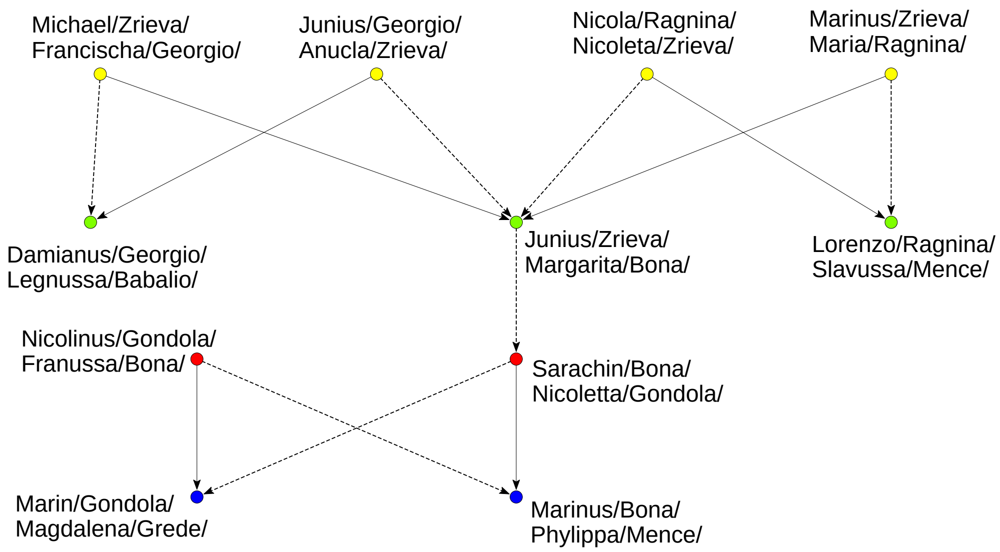


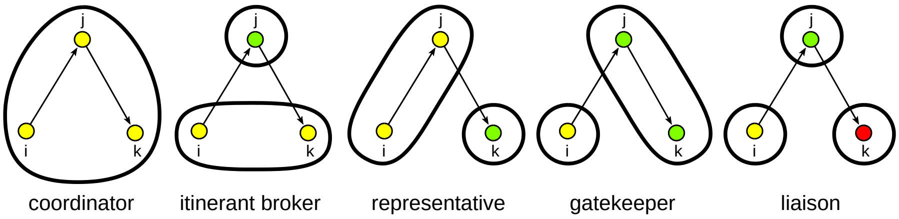
Figure 6: Fragments – Marriages among relatives in Ragusa

2.4 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).
  - **Cluster** – Extract sub-network according to selected cluster.
  - **to GEDCOM** – Extract sub-genealogy according to selected partition (for example one 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*** – 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 [6]. 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.

$$
\begin{aligned}
d_1(u,v) &= \frac{|N_u + N_v|}{\text{1st maxdegree} + \text{2nd maxdegree}} \\
d_2(u,v) &= \frac{|N_u + N_v|}{|N_u \cup N_v|} \\
d_3(u,v) &= \frac{|N_u + N_v|}{|N_u| + |N_v|} \\
d_4(u,v) &= \frac{\max(|N_u \setminus N_v|, |N_v \setminus N_u|)}{\max(|N_u|, |N_v|)} \\
d_5(u,v) &= \sqrt{\sum_{\substack{s=1 \\ s \neq u,v}}^{n} ((q_{us} - q_{vs})^2 + (q_{su} - q_{sv})^2) + p \cdot ((q_{uu} - q_{vv})^2 + (q_{uv} - q_{vu})^2)} \\
d_6(u,v) &= \sum_{\substack{s=1 \\ s \neq u,v}}^{n} (|q_{us} - q_{vs}| + |q_{su} - q_{sv}|) + p \cdot (|q_{uu} - q_{vv}| + |q_{uv} - q_{vu}|)
\end{aligned}
$$

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.

- **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 [15, 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

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

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

    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 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 arcs according to given Cluster
  - ∗ **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.
- **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. 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 [21].

- **Blockmodeling*** – Generalized blockmodeling [4, 23]. See also: Manual for MODEL 2 (description of models in input MDL files).

  – **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.

- **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):

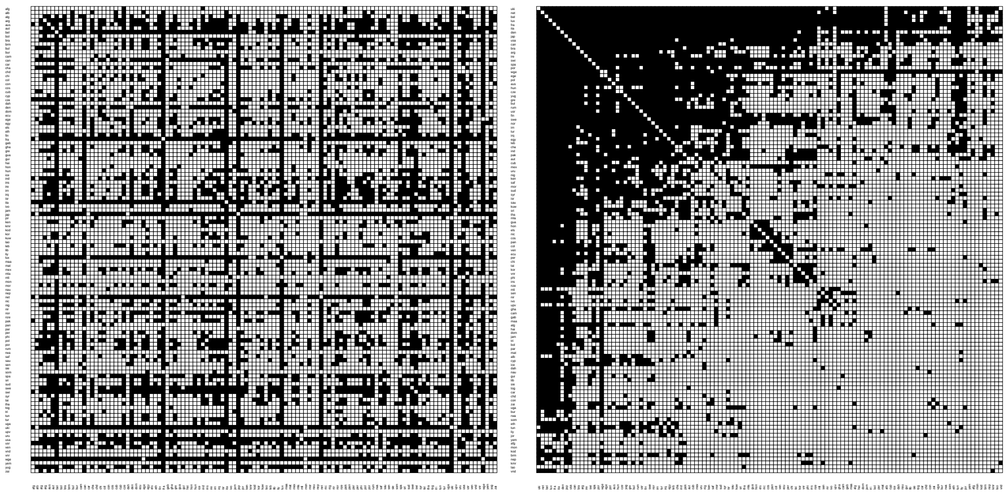


Figure 7: World trade. Orderings: alphabetical and determined by clustering

    * **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 [38, 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 [38, 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.

- **Expand Partition** – Expand partial partition (some class numbers are uknown – 0) to vertices with uknown class number:

- **Greedy Partition** – Put vertices 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.

• **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:

  - **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 [41, page 21] or `petri52` (see Figure 8) 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.

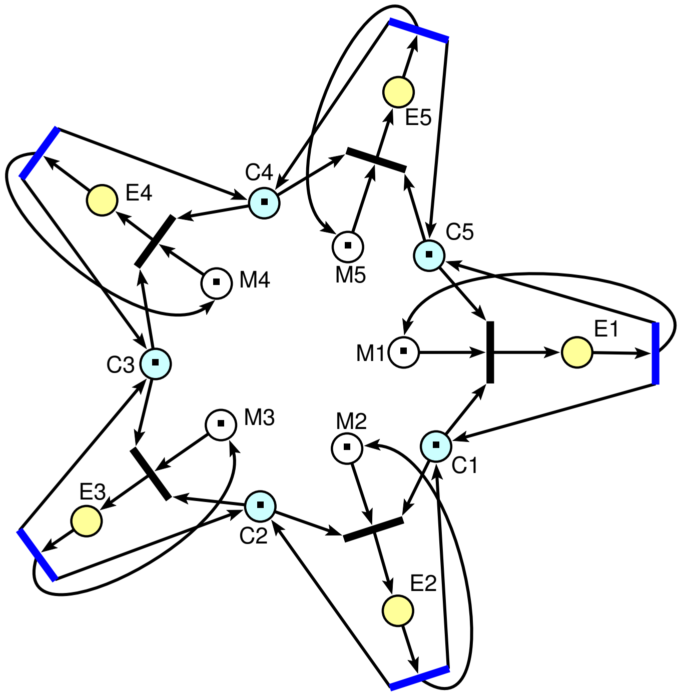


Figure 8: Petri net

2.5 Partition

Only Partition is needed as input.

- **Create Null Partition** – Create null partition of selected dimension. Default dimension is the size of selected network (if there is one in memory).
- **Binarize** – Make binary (0-1) partition from selected partition.
- **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 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.
- **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.

2.6 Partitions

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

- **First Partition** – Partition that is currently active, will be used as first (original) partition in operations.

- **Second Partition** – Partition that is currently active, will be used as second (criterion) partition in 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).
- **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 Current into First according to Second (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.
- **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.

2.7 Permut.

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.

2.8 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.

2.9 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).

2.10 Vector

Operations using vector.

- **Create Identity Vector –** Create identity vector (vector containing only 1's) 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

2.11 Vectors

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

- **First Vector –** Vector that is currently active, will be used as first vector in operations.
- **Second Vector** – Vector that is currently active, will be used as second vector in 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.
- **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.

2.12 Options

- **Read / Write**
  - **Threshold –** Value of line must be higher (absolutely) than given 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/Vertices Labels from File`.
- **Save coordinates of vertices?** – Save coordinates of vertices to network file (or not).
- **Auto Report?** – Automatically report all text results to file rep1.rep.
- **Ore: 1-Male, 2-Female links** – When reading genealogy as Ore graph generate 2 different types of arcs: arc with value 1 represents father to child relation, arc with value 2 represents mother to child relation.
- **GEDCOM – Pgraph** – Use pgraph format (nodes are couples or individuals) when reading genealogies (D. R. White), otherwise nodes are only individuals.
- **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

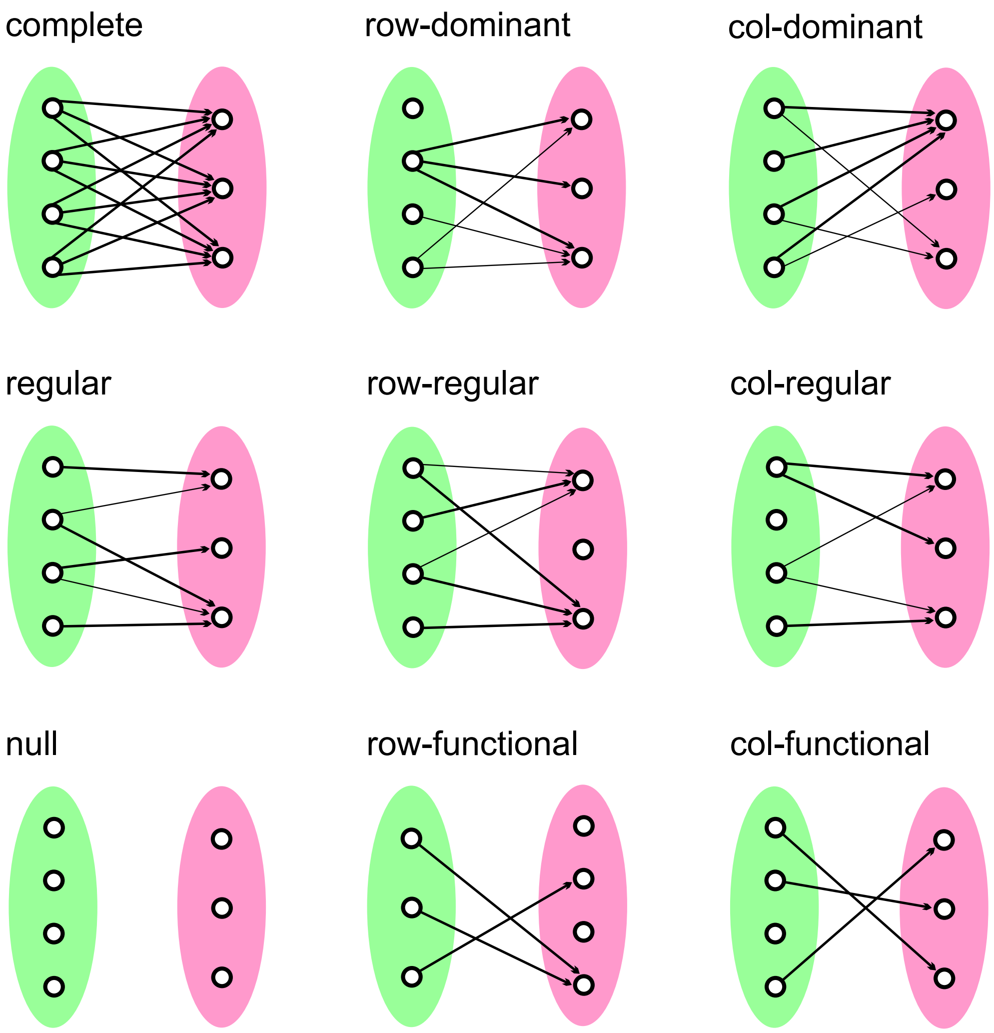


Figure 9: Generalized block types

  - 10..Degree Density

  Look in Batagelj [4] and Doreian, Batagelj, Ferligoj [23].

- **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.

2.13 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 [49] and Figure 10 on page 43.
- **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…

2.14 Tools

- **R**
  - **Send to R** – Call statistical package R [42] 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.
- **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.

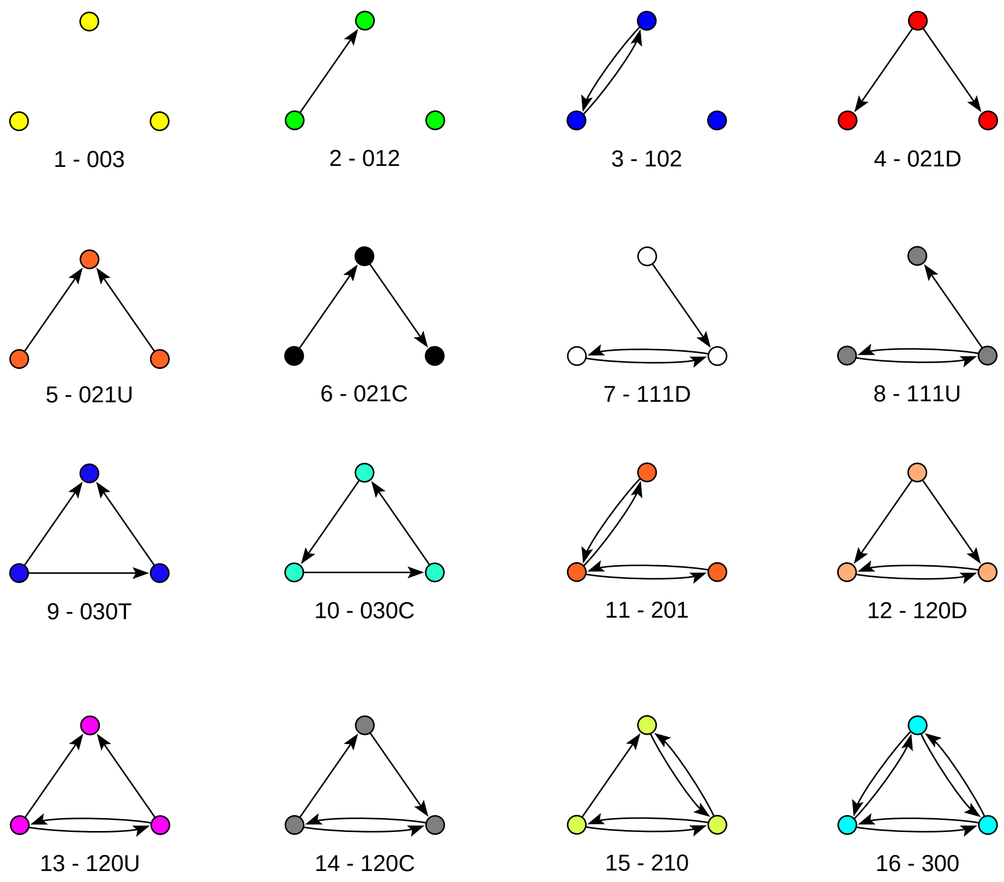


Figure 10: All different triads.

3 Draw Window Tools

3.1 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 14 on page 69 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-Partition-Vector** – Colors of vertices are determined using selected partition, sizes of vertices are determined using selected vector.

- **Draw-Vector** – Sizes of vertices are determined using selected vector.

- **Draw-SelectAll** – Create null partition and draw network using it.

3.2 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. **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
   (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)

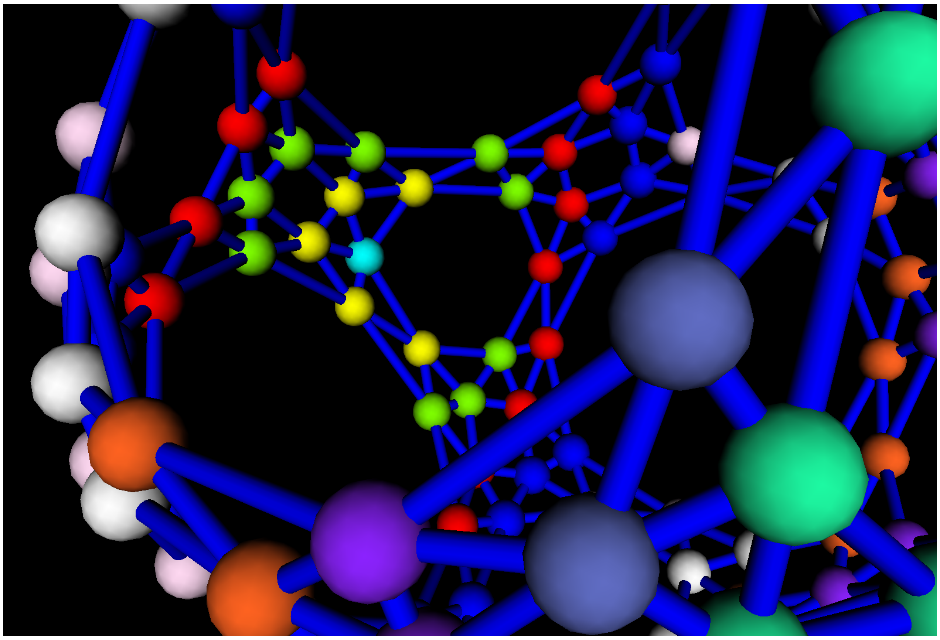


Figure 11: VRML display of layout determined by eigenvectors

3.3 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. Iterativelly 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.

3.4 GraphOnly

Show complete picture without labels and arrows.

3.5 Previous

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

3.6 Redraw

Redraw network.

3.7 Next

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

3.8 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. vector values (if vector of the same size is also selected)
  4. partition clusters (if partition of the same size is also selected)

5. without labels
6. without labels and arrows
7. parameters selected above but using real sizes (x_fact and y_fact in input files).
8. 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
  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. Size of vertices can be set to autosize (average) as well.

- **Colors** – Determine color of background, vertices, border of vertices, lines and font (of labels of vertices and lines) or turn them to default values. You can select which color should represent given class. 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 13, page 68).

  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 during moving** – Redraw whole network during moving selected vertex (slower) or not (faster).
  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).

3.9 Export

Picture to one of the following formats

- **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. The plugin for examining layouts can be obtained from:
  `http://www.adobe.com/svg/viewer/install`
  The picture in SVG can be further edited using
  `http://www.jasc.com` (Web Draw).
  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. **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.

- **VRML** – Export to VRML (Virtual Reality) format.

- **MDL MOL file** – Export to MDL Molfile format. You need Chime plugin for Netscape to watch it.
  `http://www.mdli.com` (Chemscape Chime).

- **Kinemages** – Export to Kinemages format with balls or labels. You need Mage viewer to watch it. A free copy of the Mage software can be downloaded from
  `ftp://kinemage.biochem.duke.edu/PCprograms/Win95_98_2000/`
  or `http://www.prosci.org/Kinemage/MageSoftware.html`.

  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.

- **Bitmap** – Export to Windows bitmap (bmp) format.
- **Options** – EPS, SVG and VRML default options (see section on Exports to EPS/SVG/VRML).

3.10 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.

3.11 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**

3.12 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!

4 Exports to EPS/SVG/VRML

4.1 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.

4.2 Parameters in EPS, SVG 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 13, page 68). 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*).

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*).
- **Sort Lines** – sort lines according to their values before exporting to EPS / SVG in ascending or descending order (e.g., to put lines with the highest values 'on the top' and made them in this way visible).
- **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).

Top frame on the right

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

- **EPS, SVG, VRML Size of Vertices** – default size of vertices when exporting to VRML (valid for EPS and SVG exports as well).
- **VRML Size of Lines** – width of lines in VRML.
- **VRML Bckg. Color** – background color of layout in VRML.

Middle frame on the right

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

- **Add. Border (Right, Left, Top, Bottom)** – additional border around layout when exporting to EPS (only when *EPS Clip* format is selected)
- **Background Border:**
  - **Bckg. Color** – color of background. It is used for export to SVG as well. *No* – means without background color.
  - **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.
- **Shapes file** – default shapes file, double click to change it.

Bottom frame on the right – *SVG Default*

This frame defines gradients (continuously smooth color transitions from one color to another) when we export layouts to SVG:

- **Bckg. Color 2 –** the second color for SVG export. First color is specified in the middle frame. *No* – means without second color, otherwise selected gradient will be used.
- **Bckg. Color 3 –** the third color for SVG export – gradient.
- **3D Effect on Vertices –** if selected, gradient will be applied to get 3D look of vertices.
- **Gradients –** type of gradient to use (*No*, *Linear*, or *Radial*).

4.3 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 $(1, 2, 3 \dots n)$
   - 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, ...)
     - **ic** – interior color of vertex. See Figure 13, page 68 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 alpha1, k1, alpha2, k2:

- $alpha1 = alpha2 = 0, k1 \geq 0, k2 \geq 0$ – straight line (default)
- $alpha1 = alpha2 = 0, k1 = -1, k2 > 0$ – oval edge with radius $k2$ (measure of radius is absolute as explained above)
- $alpha1 = alpha2 = 0, k1 = -1, k2 < 0$ – second possible oval edge with radius $-k2$
- $alpha1 = alpha2 = 0, k1 = -2, k2 > 0$ – circular arc with radius $k2$ in positive direction
- $alpha1 = alpha2 = 0, k1 = -2, k2 < 0$ – second possible circular arc with radius $-k2$ in positive direction
- $alpha1 = alpha2 = 0, k1 = -3, k2 > 0$ – circular arc with radius $k2$ in negative direction
- $alpha1 = alpha2 = 0, k1 = -3, k2 < 0$ – second possible circular arc with radius $-k2$ in negative direction
- $alpha1 = alpha2 = 0, k1 = -4$ – double edge
- $alpha1$ or $alpha2 \neq 0$, $k1 > 0$, $k2 > 0$ – Bezier curve (if $alpha1$ and $alpha2$ have different signs line goes from one to another side of straight line connecting both vertices, if $alpha1$ and $alpha2$ 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 LaTeX).

- 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 l "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 l "Bezier arc" lphi 90 la 0 lp 0.75
 4 2 -1 l "Straight dotted arc" p Dots c Red
*Edges
 1 3  1 l "Straight edge" lp 0.4
 3 4  1 l "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.

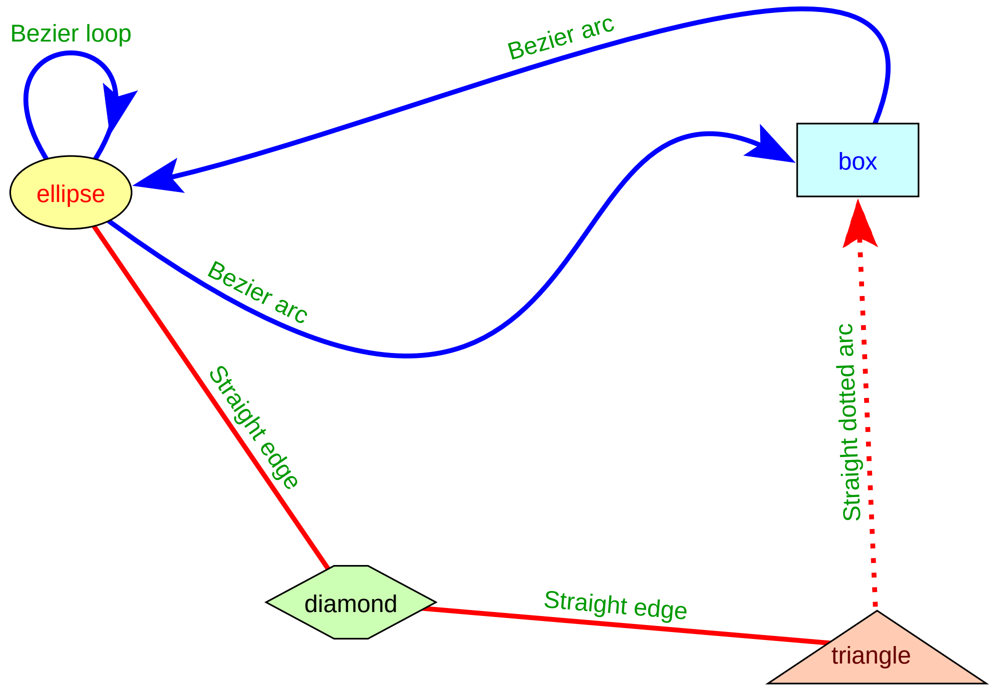


Figure 12: Example picture

5 Using Macros in Pajek

5.1 What is a Macro?

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.

5.2 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 choose Macro/Record again to stop recording.

5.3 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.

5.4 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)
```

First seven sentences 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.

5.5 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 subnetworks 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.

6 Colors in Pajek

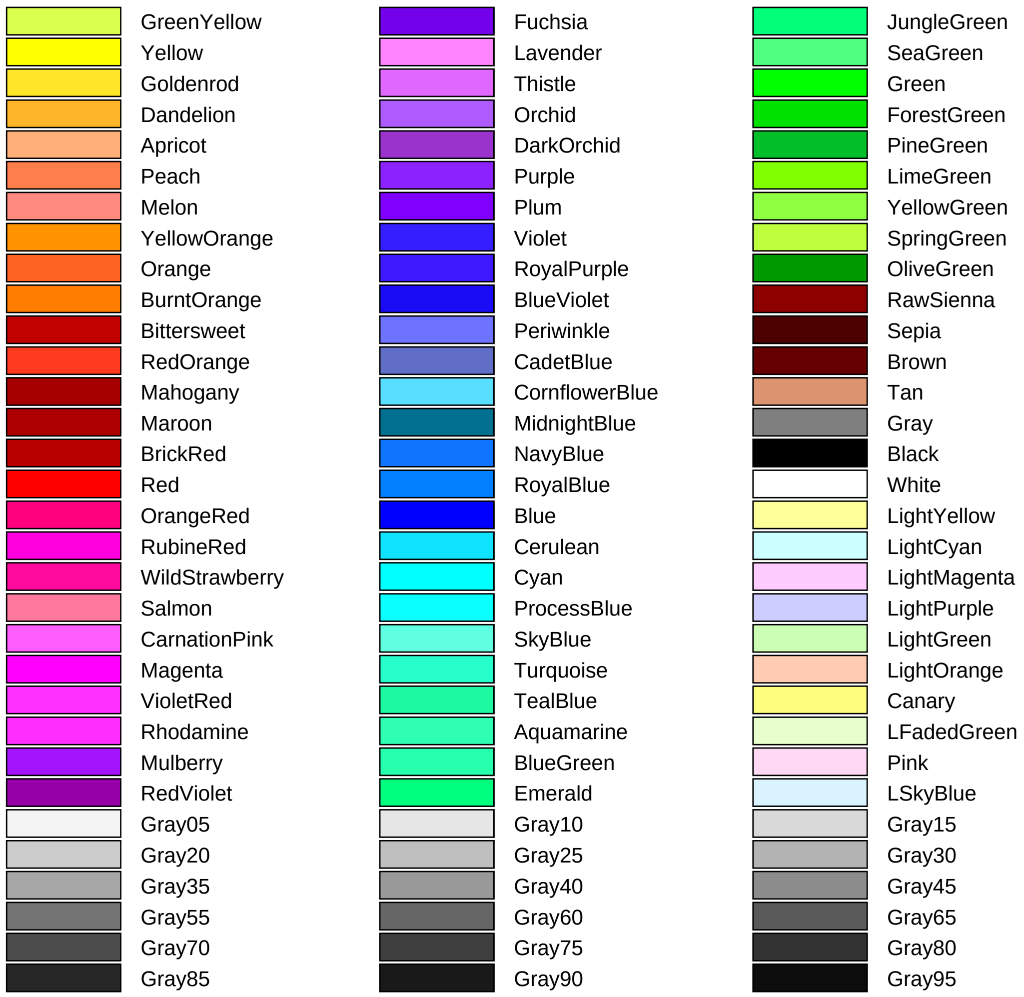


Figure 13: Colors in **Pajek**.

Figure 14: Partition colors.

7 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/**

- 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 15: Photo: Stefan Ernst, ***Gartenkreuzspinne* / *Araneus diadematus***

References

[1] Adobe SVG viewer. `http://www.adobe.com/svg/`

[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. (1997) Notes on blockmodeling. *Social Networks* **19**, 143-155.

[5] Batagelj V.: Efficient Algorithms for Citation Network Analysis.
`http://arxiv.org/abs/cs.DL/0309023`

[6] Batagelj, V., Ferligoj, A., and Doreian, P. (1992), Direct and Indirect Methods for Structural Equivalence, *Social Networks*, **14**, 63–90.

[7] Batagelj, V., Doreian, P., and Ferligoj, A. (1992) An Optimizational Approach to Regular Equivalence. *Social Networks* **14**, 121-135.

[8] Batagelj V., Mrvar A.: Pajek.
`http://vlado.fmf.uni-lj.si/pub/networks/pajek/`

[9] Batagelj V., Mrvar A. (2000) Some Analyses of Erdős Collaboration Graph. Social Networks, **22**, 173-186

[10] Batagelj V., Mrvar A. (2001) A Subquadratic Triad Census Algorithm for Large Sparse Networks with Small Maximum Degree. Social Networks, **23**, 237-243

[11] Batagelj V., Mrvar A., Zaveršnik M. (1999) Partitioning Approach to Visualization of Large Graphs. In: Kratochvil J. (Ed.) GD'99, Štiřin Castle, Czech Republic. LNCS **1731**. Springer-Verlag, 90-97.

[12] 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`

[13] Batagelj V., Zaveršnik M. (2002): Generalized Cores.
`http://arxiv.org/abs/cs.DS/0202039`

[14] Batagelj, V. and Zaveršnik, M. (2003): Short Cycles Connectivity.
`http://arxiv.org/abs/cs.DS/0308011`

[15] Bollobas B.: Random Walks on Graphs,

[16] Broder A. etal. (2000): Graph structure in the web. http://www.almaden.ibm.com/cs/k53/www9.final/

[17] Burt R.S. (1992): *Structural Holes*. The Social Structure of Competition. Cambridge MA: Harvard University Press.

[18] Butts, C.T. (2002) sna: Tools for Social Network Analysis. http://cran.at.r-project.org/src/contrib/PACKAGES.html#sna

[19] Cosmo Player (2002) http://ca.com/cosmo/

[20] de Nooy W., Mrvar A., Batagelj V. (2002) *Exploratory Social Network Analysis With* **Pajek**. to be published by the Cambridge University Press.

[21] Doreian P., Mrvar A. (1996) A Partitioning Approach to Structural Balance. Social Networks, **18**. 149-168

[22] Doreian, P., Batagelj, V., Ferligoj, A. (2000) Symmetric-acyclic decompositions of networks. *J. classif.*, **17**(1), 3-28.

[23] Doreian, Batagelj, Ferligoj: Generalized Blockmodeling. to be published by the Cambridge University Press.

[24] 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).

[25] GEDCOM 5.5. http://homepages.rootsweb.com/˜pmcbride/gedcom/55gctoc.htm

[26] Ghostscript, Ghostview and GSview. http://www.cs.wisc.edu/˜ghost/

[27] Gibbons A. (1985) Algorithmic Graph Theory. Cambridge University Press.

[28] Grossman J. (2002) The Erdős Number Project. http://www.oakland.edu/˜grossman/erdoshp.html

[29] 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/

[30] Hummon, N.P., Doreian, P. (1989) Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**, 39–63.

[31] Jones B. (2002). Computational geometry database.
`http://compgeom.cs.uiuc.edu/~jeffe/compgeom/biblios.html`

[32] Kleinberg J. (1998) Authoritative sources in a hyperlinked environment. In Proc 9th ACMSIAM Symposium on Discrete Algorithms, p. 668-677.
`http://www.cs.cornell.edu/home/kleinber/auth.ps`

[33] Knuth, D. E. (1993) *The Stanford GraphBase*. Stanford University, ACM Press, New York. `ftp://labrea.stanford.edu/pub/sgb/`

[34] LOCKS: CRA Analyses of News Stories on the Terrorist Attack. Arizona State University. `http://locks.asu.edu/terror/`

[35] 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`

[36] MDL Information Systems, Inc. (2002) `http://www.mdli.com/`

[37] James Moody home page (2002)
`http://www.soc.sbs.ohio-state.edu/jwm/`

[38] Murtagh, F. (1985) Multidimensional Clustering Algorithms, *Compstat lectures*, **4**, Vienna: Physica-Verlag.

[39] Pajek's datasets:
`http://vlado.fmf.uni-lj.si/pub/networks/data/`

[40] D.M. Pennock etal. (2002) Winners dont't take all, PNAS, **99**/8, 5207-5211.

[41] Peterson J. L.: *Petri Net Theory and the Modeling of Systems*.

[42] The R Project for Statistical Computing. `http://www.r-project.org/`

[43] Richardson D.C., Richardson J.S. (2002) The Mage Page.
`http://kinemage.biochem.duke.edu/index.html`

[44] Scott, J. (2000) Social Network Analysis: A Handbook, 2nd edition. London: Sage Publications.

[45] Seidman S. B. (1983) Network structure and minimum degree, *Social Networks*, **5**, 269–287.

[46] Tarjan, R. E. (1983) *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics Philadelphia, Pennsylvania.

[47] UCINET (2002) http://www.analytictech.com/

[48] The United States Patent and Trademark Office. http://patft.uspto.gov/netahtml/srchnum.htm

[49] Wasserman S., Faust K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge.

[50] 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

[51] Wilson, R.J., Watkins, J.J. (1990) *Graphs: An Introductory Approach*. New York: John Wiley and Sons.

[52] W3C SVG page. http://www.w3.org/Graphics/SVG