Pajek

Kumar Ashish (04CS1016)

May 6, 2008

Introduction

Pajek is a program, for Windows, for analysis and visualization of large networks having some thousands or even millions of vertices. In Slovenian language the word pajek means spider.

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

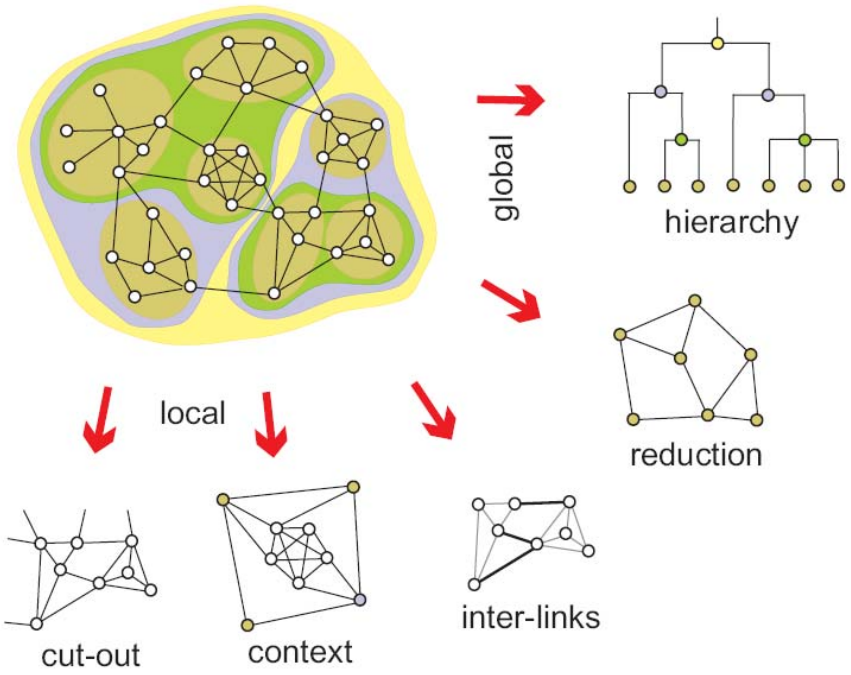
The main goals in the design of Pajek are:

1. To support abstraction by (recursive) decomposition of a large network into several smaller networks that can be treated further using more sophisticated methods.
2. To provide the user with some powerful visualization tools.
3. To implement a selection of efficient (sub quadratic) algorithms for analysis of large networks.

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

Besides ordinary (directed, undirected, mixed) networks Pajek supports also:

1. 2-mode networks, bipartite (valued) graphs - networks between two disjoint sets of vertices. Examples of such networks are: (authors, papers, cite the paper), (authors, papers, is the (co)author of the paper), (people, events, was present at), (people, institutions, is member of), (articles, shopping lists, is on the list).
2. Temporal networks, dynamic graphs - networks changing over time.



Approaches to deal with large networks

Data Objects

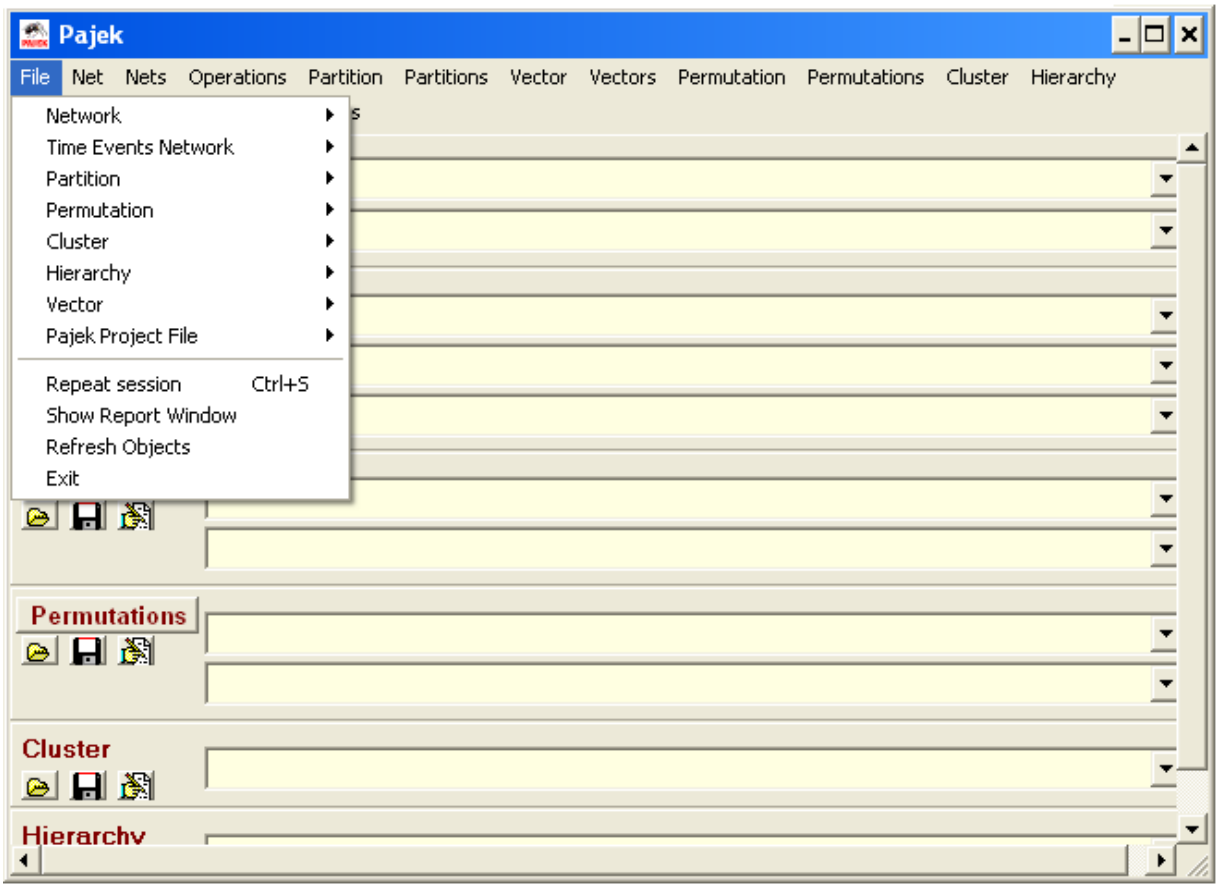
In Pajek six types of objects are used:

1. Networks: The main objects in these are vertices and lines. Default extension used in such type of file is .net.
   Network can be presented on input file in different ways:
   1. using arcs/edges (e.g. 1 2 – line from 1 to 2)
   2. using arcs lists/edges lists (e.g. 1 2 3 – line from 1 to 2 and from 1 to 3)
   3. matrix format
2. Partitions: They tell for each vertex to which class vertex it belong. Default extension used is .clu.
3. Permutations – It involves reordering of vertices. Default extension used is .per.
4. Clusters – subset of vertices (e.g. one class from partition). Default extension: .cls.
5. Hierarchies – It involves hierarchically ordered vertices. Default extension used in this case is .hie.
6. Vectors – they tell for each vertex some numerical property (real number). Default extension: .vec.

By double clicking on selected network, partition... one can show the object on screen. The procedures in Pajek's main window 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.

Main Window Tools

1. File



As is clear from the above diagram, the first option in File is Network.

1.1. Network

1. **Read** network from ASCII file.
2. **Edit** network. Choose vertex, show its neighbors and then:
    1. add new lines to/from selected vertex (by left mouse double clicking on Newline);
    2. delete lines (by left mouse double clicking);
    3. change value of line (by single right mouse clicking);

4. Subdivide line to two orthogonal lines using new invisible vertex (by single middle mouse clicking).

3. **Save** selected network to ASCII file.

The second option is Time Events Network.

1.2. Time Events Network

It reads time network described using time events.
In this the 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.

Using **Save** we can save time network in time events format.

1.3. Partition

1. **Read** partition from ASCII file.
2. **Edit** partition (put vertices to classes).
3. **Save** selected partition to ASCII file.
4. **Change Label** of selected partition.
5. **Dispose** selected partition from memory.

1.4. Permutation

1. **Read** permutation from ASCII file.
2. **Edit** permutation (interchange positions of two vertices).
3. **Save** selected permutation to ASCII file.
4. **Change** Label of selected permutation.
5. **Dispose** selected permutation from memory.

1.5. Cluster

1. **Read** cluster from ASCII file.
2. **Edit** cluster (add and delete vertices).
3. **Save** selected cluster to ASCII file.
4. **Change** Label of selected cluster.
5. **Dispose** selected cluster from memory.

1.6. Hierarchy

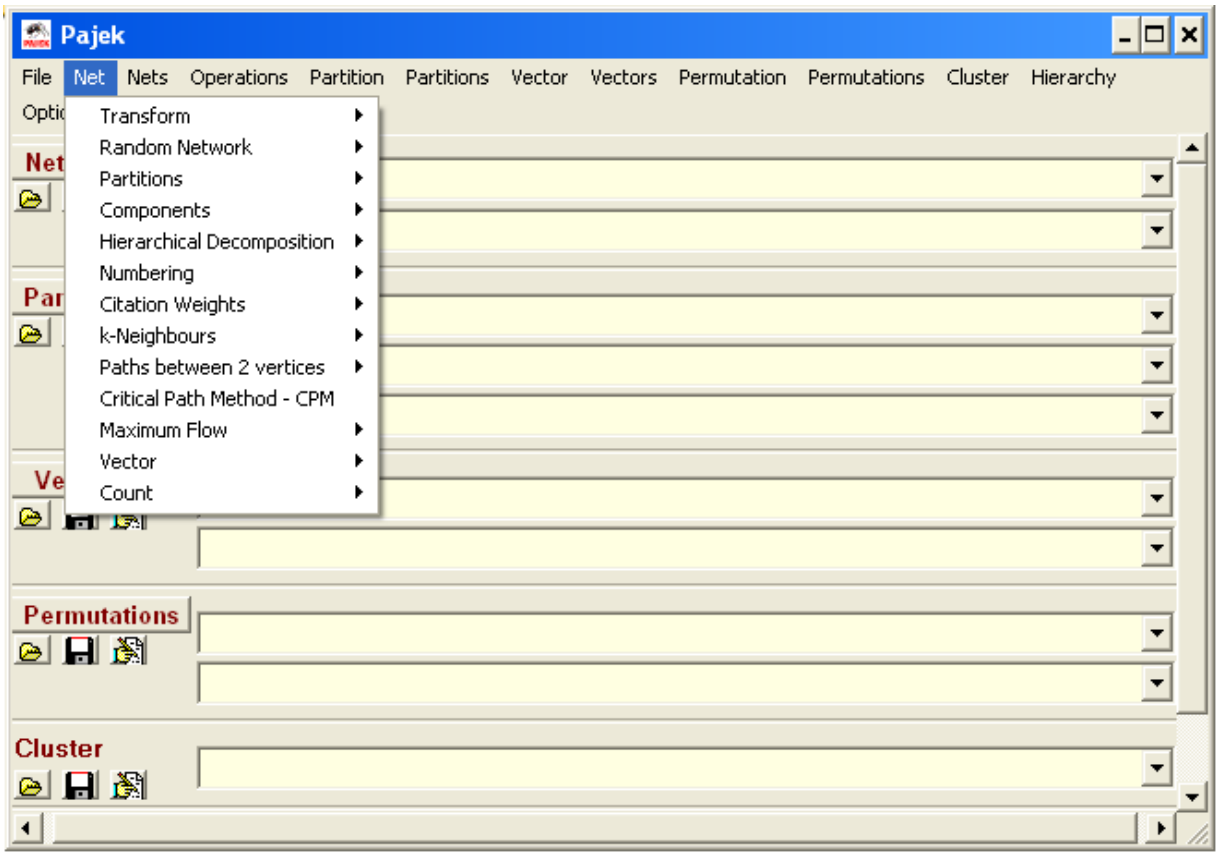
1. **Read** hierarchy from ASCII file.

2. **Edit** hierarchy (change types and names of nodes, or show vertices (and sub tree) belonging to selected node). Nodes can be pushed up and down within hierarchy.
3. **Save** selected hierarchy to ASCII file.
4. **Change** Label of selected hierarchy.
5. **Dispose** selected hierarchy from memory.

1.7. Vector

1. **Read** vector from ASCII file.
2. **Edit** vector (change components of vector).
3. **Save** selected vector(s) to ASCII file. If cluster representing vector ids 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.
4. **Change** Label of selected vector.
5. **Dispose** selected vector from memory.

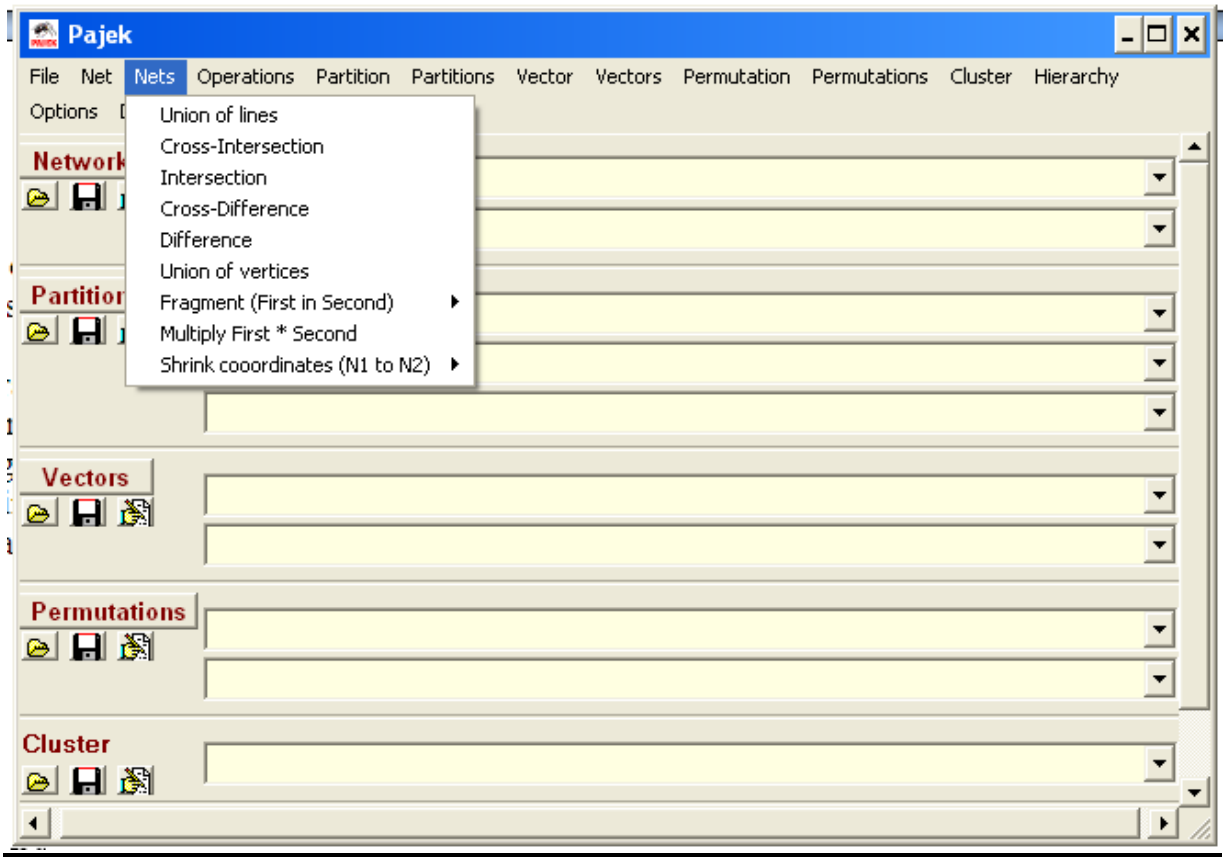
2. NET



The various operations for which only network is needed are transform, addition , conversion of edges to arcs , arcs to edges and line value reduction.

In case of transform, we have transpose in which in 1 mode we can change the direction of arrows. In 2 mode , rows and columns could be interchanged We also have removal of selected vertices , edges arcs and multiple lines according to sum values , no of lines , min value , max value and single line. Removal of loops could also be performed. Lines with value greater or lower than a specified value can also be removed.

3. Nets



It involves operation on two networks.

1. Union of lines – Fuse selected networks. Result is a multiple relations network. If you want to get union of networks, multiple lines must still be deleted. Networks must match in dimension or: If one network has m vertices and other n vertices and m < n then in network with n vertices first m vertices must match with vertices in network with m vertices.

2. Cross-Intersection – Intersection of selected networks. Networks must match in dimension or: If one network has m vertices and other n vertices and m < n then in

network with n vertices first m vertices must match with vertices in network with m vertices.

3. Intersection – Intersection of selected networks where relation numbers are taken into account.
4. Cross-Difference – Difference of selected networks.
5. Difference – Difference of selected networks where relation numbers are taken into account.
6. Union of vertices – Add the second network at the end of first network.

4. Cluster

Only cluster (and partition) is needed as input.

1. Create Empty Cluster – Create cluster without vertices.
2. Create Complete Cluster – Create cluster with values 1..n.
3. Make Partition – Transform cluster to partition.
4. 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.

5. Hierarchy

Only hierarchy is needed as input.

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

Draw Window Tools

1. Main Draw Window Tool

1. **Draw** - Draw Network. A new window is open, where a new menu appears. One 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:
2. **Draw-Partition** – Similar to Draw. Colors of vertices represent the classes in selected partition. Additionally one 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. Draw-Vector – Sizes of vertices are determined using selected vector.

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

Specifying a Network in Pajek

1. Creating Network Topology

We start off by typing "* vertices" followed by the number of vertices we wish to have in a text file. Say for example, if we want 10 vertices we will write "*vertices 10". Then vertex number is entered followed by its X , Y coordinate and in some cases Z coordinate can also be entered. These coordinates should lie in the range of 0 to 1. Some shapes can also be added to the nodes. Some of the available shapes are ellipse , diamond, box, triangle.

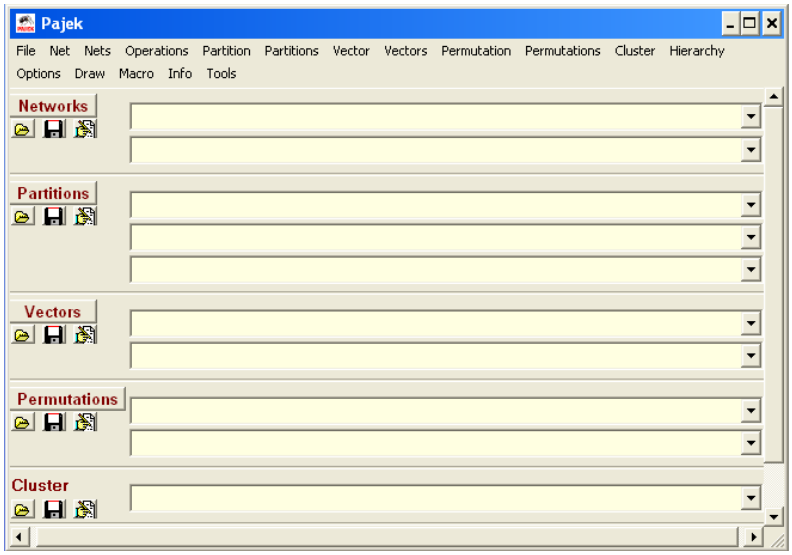
Now we may specify node relationships separately with either fancy arcs or basic vertices, but for simplicity sake's , we will use an adjacency matrix to define node relationships. Using this method we can create simple directed vertices between associated nodes. To create connection connection between nodes, 1 is placed otherwise 00 is placed.

In the end it should look something like this:-

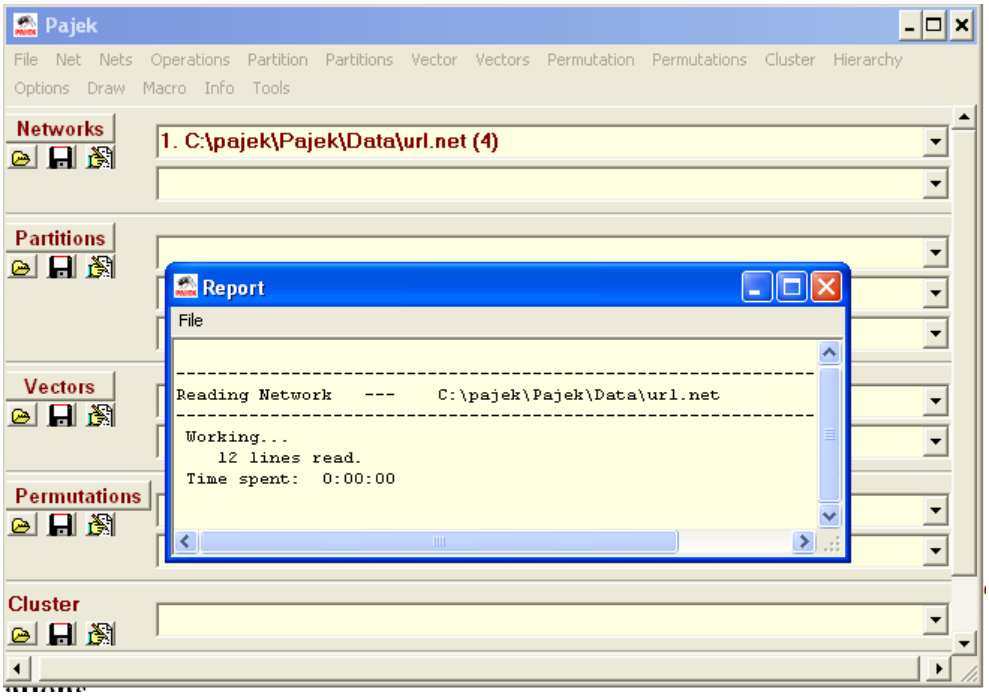
```
*vertices 4
1    "Andrej"    0.1201    0.2849    0.5000    ellipse
2    "Vlado"     0.8188    0.2488    0.5000    box
3    "Pajek"     0.3688    0.7792    0.5000    diamond
4    "Book       0.8359    0.8333    0.5000    triangle
*Edges
1 2
1 3
1 4
2 3
2 4
3 4
```

2. Pajek Execution

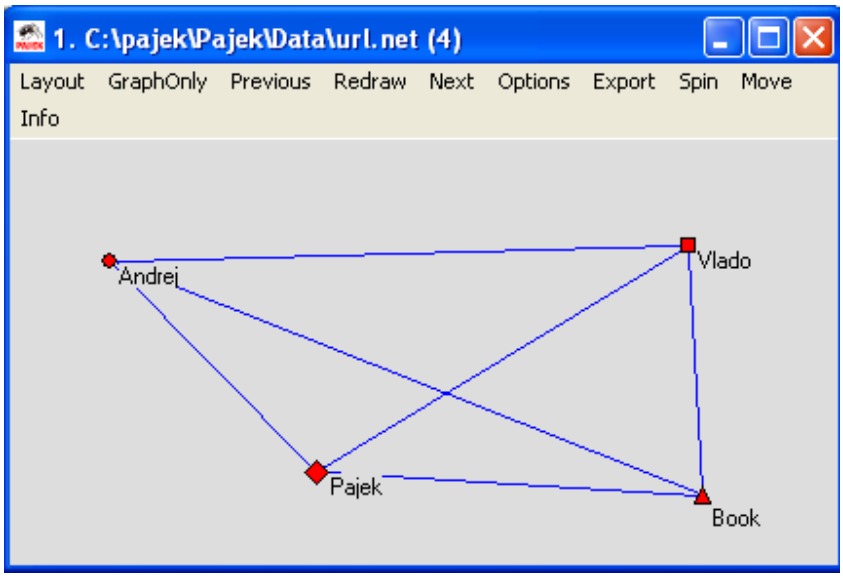
Pajek Program is started by clicking on the "Pajek" shortcut icon. The interface shown in Figure will pop up.



Then we need to specify the Pajek input 'Network' file by clicking on the yellow folder icon under 'Network'. Reading of the Network will be confirmed as shown in Figure below.



In order to display the network, one has to go to the Draw (Ctrl + G) menu option. The resulting initial layout is shown in Figure below.



Once we have got a single network we can use options of the net as it only requires network structure.

Applications

There exist several sources of large networks that are already in machine-readable form. Pajek provides tools for analysis and visualization of such networks and is applied by researchers in different areas: social network analysis, chemistry (organic molecule), biomedical/genomics research (protein-receptor interaction networks), genealogies, Internet networks, citation networks [42], diffusion networks (AIDS, news), analysis of texts, data-mining (2-mode networks) [14], etc. Although it was developed primarily for analysis of large networks it is often used also for, especially visualization of, small networks.

Algorithms

To support the design goals several algorithms known from the literature are implemented:-

Citation Weights

In a given set of units/vertices U (articles, books, works, etc.) we introduce a citing relation/set of arcs

$$R \subseteq U \times U$$

$$uRv \cong v \text{ cites } v$$

The citation network analysis started in 1964 with the paper of Garfield etal. In 1989 Hummon and Doreian [36] proposed three indices – weights of arcs that provide us with automatic way to identify the (most) important part of the citation network. For two of these indices we developed algorithms to efficiently compute them.

A citing relation is usually reflexive (no loops) and (almost) acyclic. In the following we shall assume that it has these two properties. Since in real-life citation networks the strong components are small (usually 2 or 3 vertices) we can transform such network into an acyclic network by shrinking strong components and deleting loops. For other approaches see [4]. It is also useful to transform a citation network to its standardized form by adding a common source vertex s =2 U and a common sink vertex t =2 U. The source s is linked by an arc to all minimal elements of R; and all maximal elements of R are linked to the sink t. Thus we get a st-digraph [TF 2.2]. Finally, to make the theory smoother, we add also the `feedback' arc (t; s).

The search path count (SPC) method is based on counters n(u; v) that count the number of different paths from s to t through the arc (u; v). To compute n(u; v) we introduce two auxiliary quantities: n (v) counts the number of different paths from s to v, and n+(v) counts the number of different paths from v to t.

It follows by basic principles of combinatory that:

$$n(u,v) = n^-(u).n^+(v); \quad (u,v) \in R$$

where

$$n^-(u) = \begin{cases} 1 & \text{when } u=s \\ \sum_{v:vRu} n^-(v) & \text{otherwise} \end{cases}$$

$$n^+(v) = \begin{cases} 1 \text{ when } u=t \\ \sum_{v:uRv} n^+(v) \end{cases}$$

This is the basis of an efficient algorithm for computing n(u; v) - after the topological sort [TF 2.2] of the st-digraph we can compute, using the above relations in topological order, the weights in time of order O(m), m = |R|. The topological order ensures that all the quantities in the right sides of the above equalities are already computed when needed.

The Hummon and Doreian indices are defined as follows:

1. Search path link count (SPLC) method: wl(u; v) equals the number of all possible search paths through the network emanating from an origin node" through the arc (u; v) 2 R.

2. Search path node pair (SPNP) method: wp(u; v) accounts for all connected vertex pairs along the paths through the arc (u; v) 2 R".

We get the SPLC weights by applying the SPC method on the network obtained from a given standardized network by linking the source s by an arc to each non minimal vertex from U; and

the SPNP weights by applying the SPC method on the network obtained from the SPLC network by additionally linking by an arc each non maximal vertex from U to the sink t.

The values of counters n(u; v) form a flow in the citation network – the Kirchoff's vertex law holds: For every vertex u in a standardized citation network incoming flow = outgoing flow.

$$\sum_{v:vRu} n(u,v) \ = \sum_{v:vRu} n(u,v) = n\text{-}(u).n+(v)$$

The weight n (t; s) equals to the total flow through network and provides a natural normalization of weights.

$$w(u,v) = \frac{n(u,v)}{n(t,s)} \Rightarrow 0 \le w(u,v) \le 1$$

And if C is the minimal cut set then

$$\sum w(u,v) = 1$$

In large networks the values of weights can grow very large. This should be considered in the implementation of the algorithms.

Pattern searching

If a selected pattern determined by a given graph does not occur frequently in a sparse network the straightforward backtracking algorithm applied for pattern searching finds all appearances of the pattern very fast even in the case of very large networks.

To speed up the search or to consider some additional properties of the pattern, a user can set some additional options:

1. Vertices in network should match with vertices in pattern in some nominal, ordinal or numerical property (for example, type of atom in molecule).
2. Values of edges must match (for example, edges representing male/female links in the case of p-graphs).
3. The first vertex in the pattern can be selected only from a given subset of vertices in the network.

Pattern searching was successfully applied to searching for patterns of atoms in molecule (carbon rings) and searching for relinking marriages in genealogies. Figure 5 presents three connected relinking marriages which are nonblood marriages found in the genealogy of Ragusa noble families. The genealogy is represented as a p-graph. A solid arc indicates the is a son of relation, and a dotted arc indicates the is a daughter of relation. In all three patterns a brother and a sister from one family found their partners in the same other family.