UNIVERSITY OF LJUBLJANA
INSTITUTE OF MATHEMATICS, PHYSICS AND MECHANICS
DEPARTMENT OF THEORETICAL COMPUTER SCIENCE
JADRANSKA 19, 1 000 LJUBLJANA, SLOVENIA

Preprint series, Vol. 41 (2003), 871

PAJEK
ANALYSIS AND VISUALIZATION OF LARGE NETWORKS

Vladimir Batagelj, Andrej Mrvar

ISSN 1318-4865

Version: March 4, 2003

Math.Subj.Class.(2000): 05 C 90, 68 R 10, 76 M 27, 68 U 05, 05 C 50, 05 C 85, 90 C 27, 92 H 30, 92 G 30, 93 A 15.

Supported by the Ministry of Education, Science and Sport of Slovenia, Projects J1-8532 and Z5-3350.

To be published in *Graph Drawing Software* book, edited by M. Jünger and P. Mutzel, in the Springer series *Mathematics and Visualization*.

Address: Vladimir Batagelj, University of Ljubljana, FMF, Department of Mathematics, and IMFM Ljubljana, Department of TCS, Jadranska ulica 19, 1 000 Ljubljana, Slovenia

e-mail: `vladimir.batagelj@uni-lj.si`

Ljubljana, March 14, 2003

Pajek*
Analysis and Visualization of Large Networks

Vladimir Batagelj[1] and Andrej Mrvar[2]

[1] Department of Mathematics, Faculty of Mathematics and Physics, University of Ljubljana, Slovenia
[2] Faculty of Social Sciences, University of Ljubljana, Slovenia

1 Introduction

`Pajek` is a program, for Windows, for analysis and visualization of *large networks* having some ten or houndred of thousands of vertices. In Slovenian language *pajek* means spider.

The design of `Pajek` is based on experiences gained in development of graph data structure and algorithms libraries Graph [2] and X-graph [15], collection of network analysis and visualization programs STRAN, RelCalc, Draw, Energ [9], and SGML-based graph description markup language NetML [8]. We started the development of `Pajek` in November 1996.

The main goals in the design of `Pajek` are [10,13]:

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

With `Pajek` we can (see Figure 1): *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:

- 2-mode networks, bipartite (valued) graphs – networks between two disjoint sets of vertices. Examples of such networks are: (authors, papers, *cites the paper*), (authors, papers, *is the (co)author of the paper*), (people, events, *was present at*), (people, institutions, *is member of*), (articles, shoping lists, *is on the list*).

* This work was partially supported by the Ministry of Education, Science and Sport of Slovenia, Projects J1-8532 and Z5-3350.

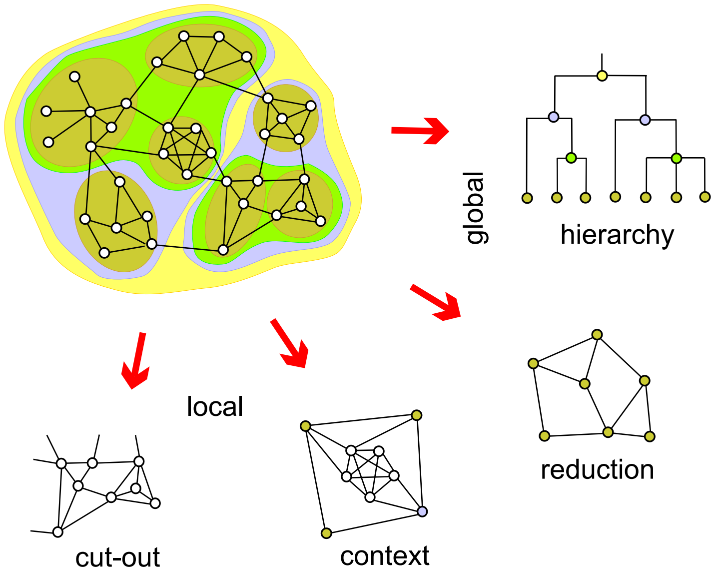


Fig. 1. *Approaches to deal with large networks*

- temporal networks, dynamic graphs – networks changing over time.

In this chapter we present the main characteristics of `Pajek`. Since large networks can't be visualized in details in a single view we have first to identify interesting substructures in such network and then visualize them as separate views. The central, algorithmic section of this chapter deals mainly with different efficient approaches to this problem.

2 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 [11], chemistry (organic molecule), biomedical/genomics research (protein-receptor interaction networks) [59], genealogies [57,28], Internet networks [22], citation networks [42], diffusion networks (AIDS, news), analysis of texts [17], 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.

In last months (end of 2002) we had over 500 downloads of `Pajek` per month.

`Pajek` is also used at several universities: Ljubljana, Rotterdam, Stanford, Irvine, The Ohio State University, Penn State, Wisconsin/Madison, Vienna, Freiburg, Madrid, and some others as a support in courses on network analysis. Together with Wouter de Nooy from University of Rotterdam we wrote a course book *Exploratory Social Network Analysis With* `Pajek`[25].

3 Algorithms

To support the design goals we implemented several algorithms known from the literature (see section 4.2), but for some tasks new, efficient algorithms, suitable to deal with large networks, had to be developed. They mainly provide different ways to identify interesting substructures in a given network.

3.1 Citation weights

In a given set of units/vertices U (articles, books, works, etc.) we introduce a *citing relation*/set of arcs $R \subseteq U \times U$

$$uRv \equiv v \text{ cites } u$$

which determines a *citation network* $N = (U, R)$.

The citation network analysis started in 1964 with the paper of Garfield et al. [29]. 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 [4].

A citing relation is usually *irreflexive* (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 $s \notin U$ and a common *sink* vertex $t \notin U$. 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 combinatorics that

$$n(u, v) = n^-(u) \cdot n^+(v), \qquad (u, v) \in R$$

where

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

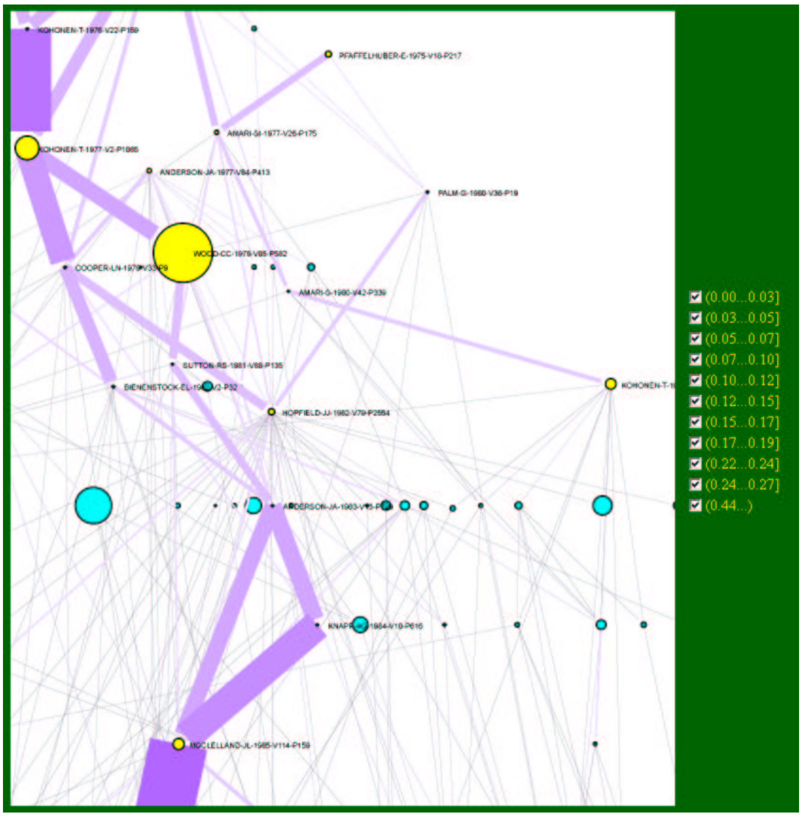


Fig. 2. Part of SOM main subnetwork at level 0.001

and

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

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:

- *search path link count* (SPLC) method: $w_l(u, v)$ equals the number of "*all possible search paths through the network emanating from an origin node*" through the arc $(u, v) \in R$.
- *search path node pair* (SPNP) method: $w_p(u, v)$ "*accounts for all connected vertex pairs along the paths through the arc* $(u, v) \in R$".

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

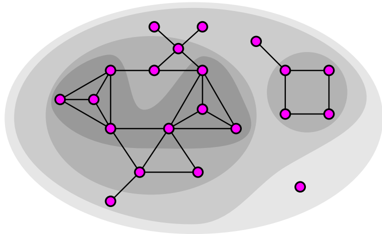


Fig. 3. *0, 1, 2 and 3 core*

to each nonminimal 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 nonmaximal 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*:

$$\sum_{v:vRu} n(v, u) = \sum_{v:uRv} n(u, v) = n^-(u) \cdot n^+(u)$$

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

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

and if C is a minimal arc-cut-set

$$\sum_{(u,v) \in C} w(u, v) = 1$$

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

In Figure 2 the main subnetwork obtained as an edge-cut at level 0.001 of the citation network ($n = 4470$, $m = 12731$) on SOM (*self-organizing maps*) literature is presented. The picture is exported in SVG with additional Javascript support that provides the user with options to inspect the subnetwork at different predetermined levels.

3.2 Cores and generalized cores

The notion of core was introduced by Seidman in 1983 [51]. Let $G = (V, E)$ be a graph. A subgraph $H = (W, E|W)$ induced by the set W is a *k-core* or a *core of order k* iff $\forall v \in W : \deg_H(v) \geq k$, and H is a maximal subgraph with this property. The core of maximum order is also called the *main* core. The

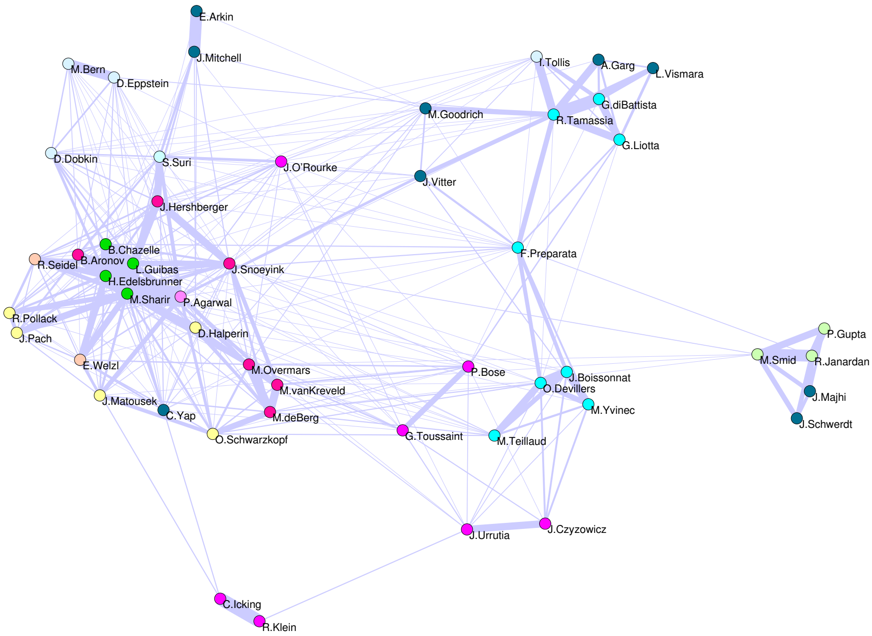


Fig. 4. p_S-core at level 46 of Geomlib network

core number of vertex v is the highest order of a core that contains this vertex. The degree $\deg(v)$ can be: in-degree, out-degree, in-degree + out-degree, etc., determining different types of cores.

In Figure 3 an example of cores decomposition of a given graph is presented. From this figure we can see the following properties of cores:

- The cores are nested: $i < j \implies H_j \subseteq H_i$
- Cores are not necessarily connected subgraphs.

Our algorithm for determining the cores hierarchy is based on the following property [16]:

> If from a given graph $G = (V, E)$ we recursively delete all vertices, and edges incident with them, of degree less than k, the remaining graph is the k-core.

Its outline is given in Algorithm 1. In the refinements of the algorithm we have to provide efficient implementations of sorting the *degree*s and their reordering. Since the values of degrees are in the range $0..n-1$ we can order them in $O(n)$ using a variant of bin sort; and the update of the ordering can be done in a constant time. For details see [18].

The cores, because they can be determined very efficiently, are one among few concepts that provide us with meaningful decompositions of large networks. We expect that different approaches to the analysis of large networks

Algorithm 1: Core Numbers Algorithm

Input : Graph $G = (V, E)$ represented by lists of neighbors
Output : Table $core[V]$ with core number for each vertex

```
Compute the degrees of vertices
Order the set of vertices V in increasing order of their degrees
for each v ∈ V in the order do
    Set core[v] = degree[v]
    for each u ∈ adj(v) do
        if degree[u] > degree[v] then
            Set degree[u] = degree[u] − 1
            Reorder V accordingly
        end
    end
end
```

can be built on this basis. For example: we get the following bound on the chromatic number of a given graph G

$$\chi(G) \leq 1 + \mathrm{core}(G)$$

Cores can also be used to localize the search for interesting subnetworks in large networks since: if it exists, a k-component is contained in a k-core; and a k-clique is contained in a k-core.

The notion of core can be generalized to networks. Let $N = (V, E, w)$ be a network, where $G = (V, E)$ is a graph and $w : E \to \mathbb{R}$ is a function assigning values to edges. A *vertex property function* on $\mathbf{N}$, or a *p-function* for short, is a function $p(v, U)$, $v \in V$, $U \subseteq V$ with real values. Let $\mathrm{adj}_U(v) = \mathrm{adj}(v) \cap U$. Besides degrees, here are some examples of p-functions:

$$p_S(v, U) = \sum_{u \in \mathrm{adj}_U(v)} w(v, u), \text{ where } w : E \to \mathbb{R}_0^+$$
$$p_M(v, U) = \max_{u \in \mathrm{adj}_U(v)} w(v, u), \text{ where } w : E \to \mathbb{R}$$
$$p_k(v, U) = \text{ number of cycles of length } k \text{ through vertex } v \text{ in } (U, E|U)$$

The subgraph $H = (C, E|C)$ induced by the set $C \subseteq V$ is a *p-core at level* $t \in \mathbb{R}$ iff $\forall v \in C : t \leq p(v, C)$ and C is a maximal such set.

The function p is *monotone* iff it has the property

$$C_1 \subset C_2 \Rightarrow \forall v \in V : (p(v, C_1) \leq p(v, C_2))$$

The degrees and the functions p_S, p_M and p_k are monotone. For a monotone function the p-core at level t can be determined, as in the ordinary case, by successively deleting vertices with value of p lower than t; and the cores on different levels are nested

$$t_1 < t_2 \Rightarrow H_{t_2} \subseteq H_{t_1}$$

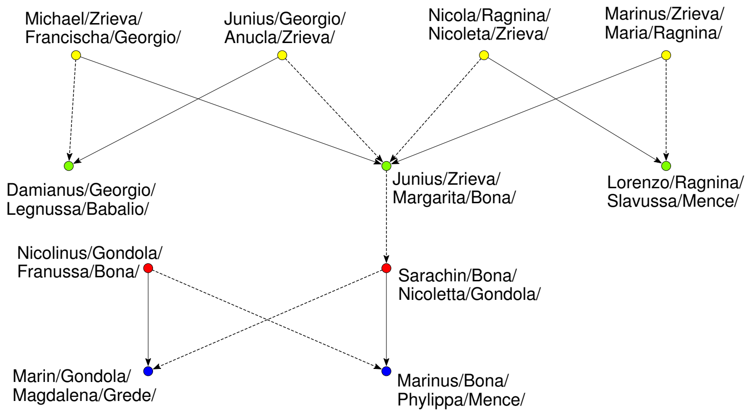


Fig. 5. Marriages among relatives in Ragusa

The p-function is *local* iff

$$p(v, U) = p(v, \mathrm{adj}_U(v))$$

The degrees, p_S and p_M are local; but p_k is **not** local for $k \geq 4$. For a local p-function an $O(m \max(\Delta, \log n))$ algorithm for determining the p-core levels exists, assuming that $p(v, \mathrm{adj}_C(v))$ can be computed in $O(\deg_C(v))$ [19].

In Figure 4 a p_S-core at level 46 of the *collaboration network* in the field of computational geometry [37] is presented.

3.3 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 appearences 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:

- vertices in network should match with vertices in pattern in some nominal, ordinal or numerical property (for example, type of atom in molecula);
- values of edges must match (for example, edges representing male/female links in the case of p-graphs [57]);
- 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 molecula (carbon rings) and searching for relinking marriages in genealogies. Figure 5 presents three connected relinking marriages which are non-blood marriages found in the genealogy of ragusan noble families [28]. The

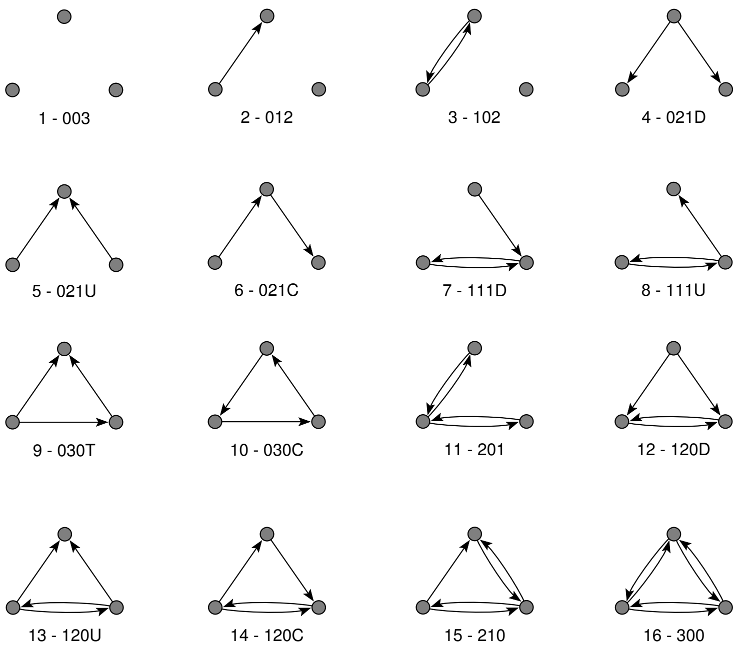


Fig. 6. Triads

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.

3.4 Triads

Let $G = (V, R)$ be a simple directed graph without loops. A *triad* is a subgraph induced by a given set of three vertices. There are 16 nonisomorphic (types of) triads [55, page 244]. They can be partitioned into three basic types (see Figure 6):

- the *null* triad 003;
- *dyadic* triads 012 and 102; and
- *connected* triads: 111D, 201, 210, 300, 021D, 111U, 120D, 021U, 030T, 120U, 021C, 030C and 120C.

Several properties of a graph can be expressed in terms of its *triadic spectrum* – distribution of all its triads. It also provides ingredients for p^* network models [56]. A direct approach to determine the triadic spectrum is of order $O(n^3)$; but in most large graphs it can be determined much faster [12]. The algorithm is based on the folllowing observation: *in a large and sparse graph most triads are null triads.* Let T_1, T_2, T_3 be the number of null, dyadic and connected triads. Since the total number of triads is $T = \binom{n}{3}$ and the above types partition the set of all triads, the idea of the algorithm is as follows:

- count all dyadic T_2 and all connected T_3 triads with their subtypes;
- compute the number of null triads $T_1 = T - T_2 - T_3$.

In the algorithm we have to assure that every non-null triad is counted exactly once while scanning the set of arcs. A set of three vertices $\{v, u, w\}$ can be in general selected in 6 different ways (v, u, w), (v, w, u), (u, v, w), (u, w, v), (w, v, u), (w, u, v). We solve the isomorphism problem by introducing the *canonical* selection that contributes to the triadic count; the other, noncanonical selections need not to be considered in the counting process.

Every connected dyad forms a dyadic triad with every vertex both members of the dyad are not adjacent to. Let $\hat{R} = R \cup R^{-1}$. Each pair of vertices (v, u), $v < u$ connected by an arc contributes

$$n - |\hat{R}(u) \cup \hat{R}(v) \setminus \{u, v\}| - 2$$

triads of type 3 – 102, if u and v are connected in both directions; and of type 2 – 012 otherwise. The condition $v < u$ determines the canonical selection for dyadic triads. A selection (v, u, w) of connected triad is canonical iff $v < u < w$.

The triads isomorphism problem can be efficiently solved by assigning to each triad a code – an integer number between 0 to 63 obtained by treating the out-diagonal entries of triad adjacency matrix as a binary number. Each triad code corresponds to a unique triad type that can be determined from a precomputed table.

For a connected triad we can always assume that v is the smallest of its vertices. So we have to determine the canonical selection from the remaining two selections (v, u, w) and (v, w, u). If $v < w < u$ and $v\hat{R}w$ then the selection (v, w, u) was already counted before. Therefore we have to consider it as canonical only if it is not $v\hat{R}w$.

In an implementation of the algorithm we must also take care about the range overflow in the case of T and T_1.

The total complexity of the algorithm is $O(\hat{\Delta} m)$ and thus, for graphs with small maximum degree $\hat{\Delta} \ll n$, since $2m \leq n\hat{\Delta}$, of order $O(n)$.

3.5 Triangular connectivities

In this subsection we present an extension of notion of connectivity to connectivity by chains of triangles.

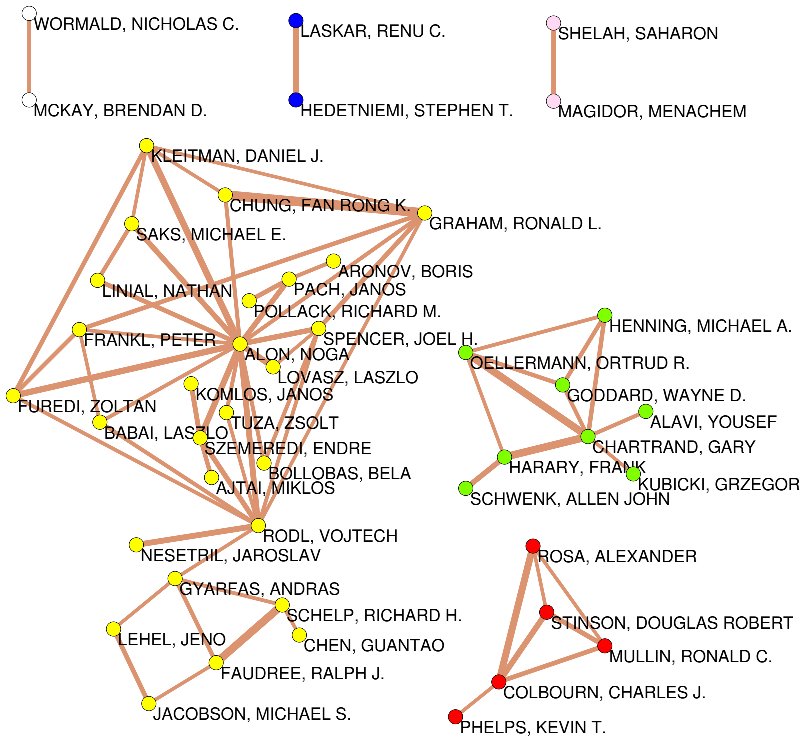


Fig. 7. Edge-cut at level 16 of triangular network of Erdős collaboration graph

Undirected graphs

We call a *triangle* a subgraph isomorphic to K_3. A subgraph $H = (V', E')$ of $G = (V, E)$ is *triangular* if each its vertex and each its edge belongs to at least one triangle in H.

A sequence $(T_1, T_2, \ldots, T_s)$ of triangles of G *(vertex) triangularly connects* vertices $u, v \in V$ iff $u \in T_1$ and $v \in T_s$ or $u \in T_s$ and $v \in T_1$ and $V(T_{i-1}) \cap V(T_i) \neq \emptyset$, $i = 2, \ldots s$. Such sequence is called a *triangular chain*. It *edge triangularly connects* vertices $u, v \in V$ iff a stronger version of the second condition holds $E(T_{i-1}) \cap E(T_i) \neq \emptyset$, $i = 2, \ldots s$.

A pair of vertices $u, v \in V$ is *(vertex) triangularly connected* iff $u = v$, or there exists a chain that triangularly connects u and v. Triangular connectivity is an equivalence relation on the set of vertices V; and nontrivial triangular connectivity components are exactly maximal connected triangular subgraphs.

A pair of vertices $u, v \in V$ is *edge triangularly connected* iff $u = v$, or there exists a chain that edge triangularly connects u and v. Edge triangular connectivity components determine an equivalence relation on the set of edges E. Each nontriangular edge is in its own component.

Let G be a simple undirected graph. A *triangular network* $N_T(G) = (V, E_T, w)$ determined by G is a subgraph $G_T = (V, E_T)$ of G which set of edges E_T consists of all triangular edges of $E(G)$. For $e \in E_T$ the weight $w(e)$ equals to the number of different triangles in G to which e belongs.

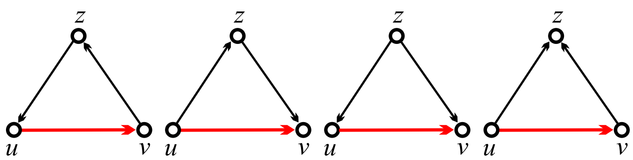
A procedure for determining E_T and $w(e), e \in E_T$ simply collects all edges with $w(e) = |\mathrm{adj}(u) \cap \mathrm{adj}(v)| > 0$, $e = \{u, v\} \in E$. If the sets of neighbors $\mathrm{adj}(v)$ are ordered we can use merging to compute $w(e)$ faster. Nontrivial triangular connectivity components are exactly the components of G_T.

Triangular networks can be used to efficiently identify dense clique-like parts of a graph. If an edge e belongs to a k-clique in G then $w(e) \geq k - 2$.

In Figure 7 the edge-cut at level 16 of triangular network of *Erdős collaboration graph* [34,11] (without Erdős, $n = 6926$, $m = 11343$) is presented.

Directed graphs

If the graph G is mixed we replace edges with pairs of opposite arcs. In the following let $G = (V, A)$ be a simple directed graph without loops. For a selected arc $(u, v) \in A$ there are four different types of directed triangles: **cyc**lic, **tra**nsitive, **in**put and **out**put.



cyc tra in out

For each type we get the corresponding triangular network N_{cyc}, N_{tra}, N_{in} and N_{out}. Also procedures for determining the networks are similar to undirected case. For example, for the cyclic network $N_{cyc} = (V, A_{cyc}, w_{cyc})$ we have for $(u, v) \in A_{cyc}$

$$w_{cyc}(u, v) = |\mathrm{outadj}(v) \cap \mathrm{inadj}(u)|$$

In directed graphs we distinguish weak and strong connectivity. The weak connectivity can be reduced to the undirected concepts in the skeleton $S = (V, E_S)$ of the given graph G

$$E_S = \{\{u, v\} : u \neq v \wedge (u, v) \in A\}$$

A subgraph $H = (V', A')$ of G is *cyclic triangular* if each its vertex and each its arc belongs to at least one cyclic triangle in H. A connected cyclic triangular subgraph is also strongly connected.

A sequence $(T_1, T_2, \ldots, T_s)$ of cyclic triangles of G *(vertex) cyclic triangularly connects* vertex $u \in V$ to vertex $v \in V$ iff $u \in T_1$ and $v \in T_s$ or $u \in T_s$ and $v \in T_1$ and $V(T_{i-1}) \cap V(T_i) \neq \emptyset$, $i = 2, \ldots s$; such sequence is called a *cyclic triangular chain*. It *arc cyclic triangularly connects* vertex u to vertex

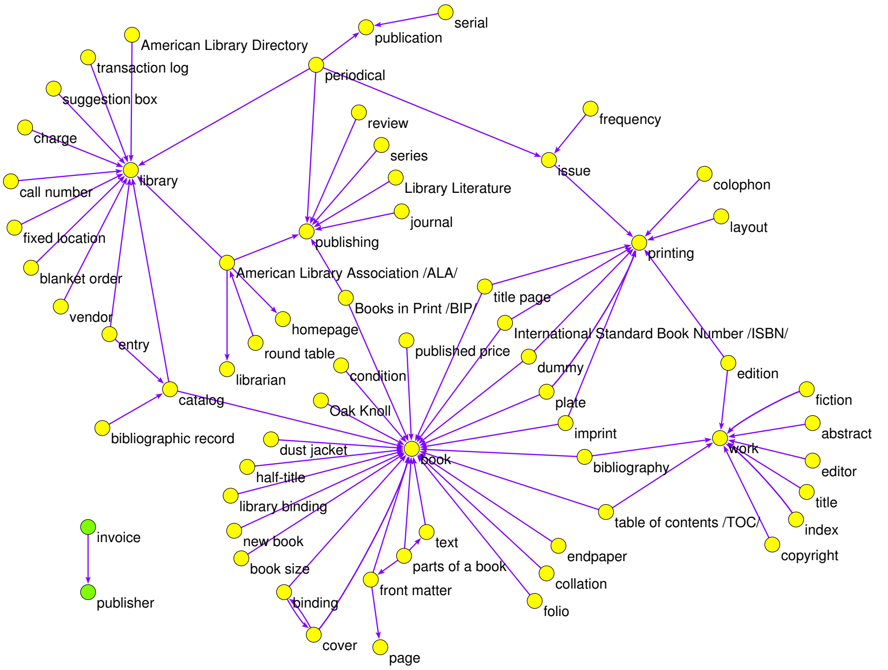


Fig. 8. Edge-cut at level 11 of transitive network of ODLIS dictionary graph

v iff $A(T_{i-1}) \cap A(T_i) \neq \emptyset$, $i = 2, \ldots s$ holds; such sequence is called an *arc cyclic triangular chain*.

Again, we can introduce two types of cyclic triangular connectivity:

A pair of vertices $u, v \in V$ is *(vertex) cyclic triangularly connected* iff $u = v$, or there exists a cyclic triangular chain that connects u to v.

A pair of vertices $u, v \in V$ is *arc cyclic triangularly connected* iff $u = v$, or there exists an arc cyclic triangular chain that connects u to v.

Cyclic triangular connectivity is an equivalence relation on the set of vertices V; and the arc cyclic triangular connectivity components determine an equivalence relation on the set of arcs A.

There exists also a parallel to unilateral connectivity. The vertex $v \in V$ is *transitively triangularly reachable* from the vertex $u \in V$ iff $u = v$, or there exists a walk from u to v in which each arc is transitive – is a base of some transitive triangle.

Transitive arcs are essentially reinforced arcs. If we remove from a graph $G = (V, A)$ a transitive arc the reachability relation in V does not change.

In Figure 8 the edge-cut at level 11 of transitive network of *ODLIS dictionary graph* [45] is presented.

These notions can be generalized to short cycle connectivity [20].

3.6 Generating large random networks

Let $p \in [0,1]$ be a given probability. An *Erdős-Rényi random graph* $G \in \mathcal{G}(n,p)$ is obtained by selecting every edge $\{u,v\}$ with a probability p:

$$\Pr(\{u,v\} \in G) = p$$

It is easy to write a program to do this:

$E = \emptyset$;
for $u = 1$ **to** $n-1$ **do for** $v = u+1$ **to** n **do**
  if random $< p$ **then** $E = E \cup \{\{u,v\}\}$;

But, for large and very sparse networks this is too slow. A faster procedure can be built on the following idea: move by random steps over the $M = \binom{n}{2}$ cells and mark the touched cells.

How to select the length of the random step? For our Bernoulli model we have $\Pr(\text{step} = s) = q^{s-1}p, \quad s = 1,2,3,\ldots$ and $F(s) = \Pr(\text{step} < s) = \sum_{t=1}^{s-1} q^{t-1}p = 1-q^{s-1}$. Therefore we get the random step s from the equation $F(s) = \text{random}$

$$s = F^{-1}(\text{random}) = 1 + \lfloor \frac{\log(1-\text{random})}{\log q} \rfloor$$

This is the basis of the fast random graph generation procedure presented in Algorithm 2. The expected number of steps of this procedure is Mp.

Algorithm 2: Sparse Erdős-Rényi random graph generator

Input : Probability p, Number of vertices n
Output : Random graph $G = (1..n, E)$

Set $q = 1-p$; $f = 1$; $u = 2$; $k = 0$; $E = \emptyset$; $M = n(n-1)/2$; $again = true$
while *again* **do**
  Set $k = k + 1 + \lfloor \frac{\ln(1-\text{random})}{\ln q} \rfloor$
  if $k > M$ **then** Set $again = false$ **else**
    while $f < k$ **do** Set $f = f + u$; $u = u + 1$
    Set $v = k + u - f - 1$; $E = E \cup \{\{u,v\}\}$
  end
od

The same approach is easy to adapt to generate different types of random graphs: undirected, directed, acyclic, undirected bipartite, directed bipartite, acyclic bipartite, 2-mode, and others [5].

`Pajek` contains also a refinement of the model for generating *scale free networks*, proposed in [47]. At each step of the growth a new vertex and k

edges are added to the network N. The endpoints of the edges are randomly selected among all vertices according to the probability

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

where $\alpha + \beta + \gamma = 1$. It is easy to check that $\sum_{v \in V} \Pr(v) = 1$. The time complexity of this procedure is $O(m)$.

3.7 2-mode networks

A 2-*mode network* is a structure $N = (U, V, A, w)$, where U and V are disjoint sets of vertices, A is the set of arcs with the initial vertex in the set U and the terminal vertex in the set V, and $w : A \to \mathbb{R}$ is a *weight*. If no weight is defined we can assume a constant weight $w(u, v) = 1$ for all arcs $(u, v) \in A$. The set A can be viewed also as a relation $A \subseteq U \times V$. A 2-mode network can be formally represented by rectangular matrix $\mathbf{A} = [a_{uv}]_{U \times V}$.

$$a_{uv} = \begin{cases} w(u, v) & (u, v) \in A \\ 0 & \text{otherwise} \end{cases}$$

For direct analysis of 2-mode networks we can use eigen-vector approach, clustering and blockmodeling. But most often we transform a 2-mode network into an ordinary (1-mode) network $N_1 = (U, E_1, w_1)$ or/and $N_2 = (V, E_2, w_2)$, where E_1 and w_1 are determined by the matrix $\mathbf{A}^{(1)} = \mathbf{A}\mathbf{A}^T$, $a^{(1)}_{uv} = \sum_{z \in V} a_{uz} \cdot a^T_{zv}$. Evidently $a^{(1)}_{uv} = a^{(1)}_{vu}$. There is an edge $\{u, v\} \in E_1$ in N_1 iff $\mathrm{adj}(u) \cap \mathrm{adj}(v) \neq \emptyset$. Its weight is $w_1(u, v) = a^{(1)}_{uv}$. The network N_2 is determined in a similar way by the matrix $\mathbf{A}^{(2)} = \mathbf{A}^T\mathbf{A}$. The networks N_1 and N_2 are analyzed using standard methods.

3.8 Normalizations

The *normalization* approach was developed for quick inspection of (1-mode) networks obtained from 2-mode networks [14,60] – a kind of network based data-mining. In networks obtained from large 2-mode networks there are often huge differences in weights. Therefore it is not possible to compare the vertices according to the raw data. First we have to normalize the network to make the weights comparable. There exist several ways how to do this. Some of them are presented in Table 1. They can be used also on other networks.

In the case of networks without loops we define the diagonal weights for undirected networks as the sum of out-diagonal elements in the row (or column)

$$w_{vv} = \sum_u w_{vu}$$

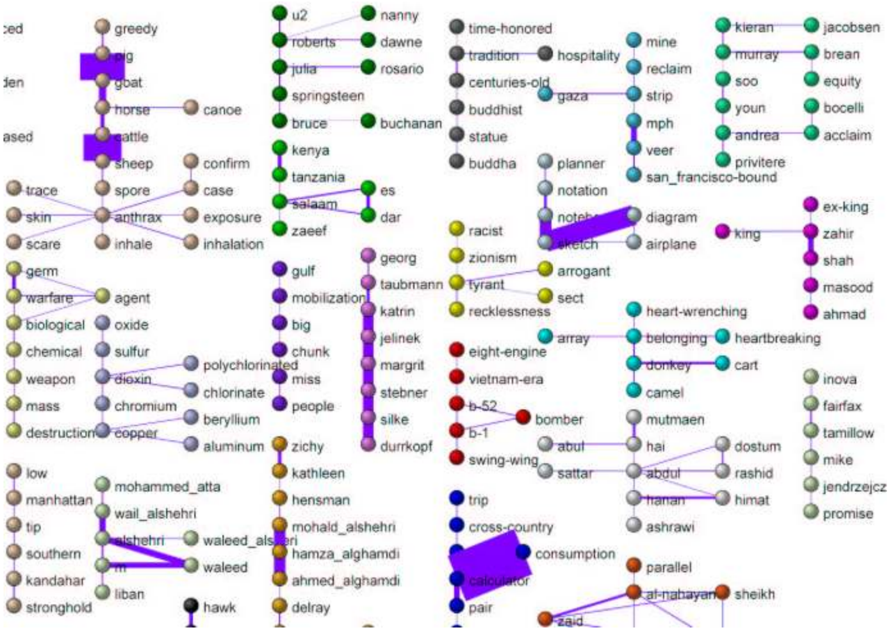


Fig. 9. GeoDeg normalization of Reuters terror news network

Table 1. Weight normalizations

$$\mathrm{Geo}_{uv} = \frac{w_{uv}}{\sqrt{w_{uu} w_{vv}}} \qquad \mathrm{GeoDeg}_{uv} = \frac{w_{uv}}{\sqrt{\deg_u \deg_v}}$$

$$\mathrm{Input}_{uv} = \frac{w_{uv}}{w_{vv}} \qquad \mathrm{Output}_{uv} = \frac{w_{uv}}{w_{uu}}$$

$$\mathrm{Min}_{uv} = \frac{w_{uv}}{\min(w_{uu}, w_{vv})} \qquad \mathrm{Max}_{uv} = \frac{w_{uv}}{\max(w_{uu}, w_{vv})}$$

$$\mathrm{MinDir}_{uv} = \begin{cases} \frac{w_{uv}}{w_{uu}} & w_{uu} \le w_{vv} \\ 0 & \text{otherwise} \end{cases} \qquad \mathrm{MaxDir}_{uv} = \begin{cases} \frac{w_{uv}}{w_{vv}} & w_{uu} \le w_{vv} \\ 0 & \text{otherwise} \end{cases}$$

and for directed networks as some mean value of the row and column sum, for example

$$w_{vv} = \frac{1}{2}\Big(\sum_u w_{vu} + \sum_u w_{uv}\Big)$$

Usually we assume that the network does not contain any isolated vertex.

After a selected normalization the important parts of network are obtained by edge-cutting the normalized network at selected level t and preserving components with at least k vertices.

In Figure 9 a part of 'themes' from *Reuters terror news network* [14] determined by a cut of its GeoDeg normalization is presented.

3.9 Blockmodeling

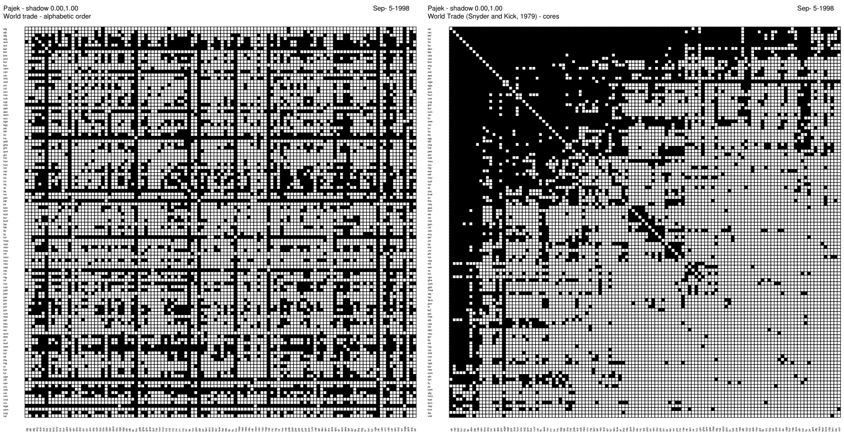


Fig. 10. Orderings

In Figure 10 the *Snyder and Kick's world trade network* is presented by its matrix: on the left side the units (states) are ordered in the alphabetic order of their names; on the right side they are ordered on the basis of clustering results. It is evident that a 'proper' ordering can reveal a structure in the network. Such orderings can be produced in different ways [44]. On the networks of moderate size (up to some hundreds of units) we can use also the blockmodeling methods.

The goal of *blockmodeling* is to reduce a large, potentially incoherent network to a smaller comprehensible structure that can be interpreted more readily [6,3,7]. One of the main procedural goals of blockmodeling is to identify, in a given network $N = (U, R)$, $R \subseteq U \times U$, *clusters* (classes) of units/vertices that share structural characteristics defined in terms of R. The units within a cluster have the same or similar connection patterns to other units. They form a *clustering* $\mathbf{C} = \{C_1, C_2, \ldots, C_k\}$ which is a *partition* of the set U. Each partition determines an equivalence relation (and vice versa).

A clustering $\mathbf{C}$ partitions also the relation R into *block*s

$$R(C_i, C_j) = R \cap C_i \times C_j$$

Each such block consists of units belonging to clusters C_i and C_j and all arcs leading from cluster C_i to cluster C_j. If $i = j$, a block $R(C_i, C_i)$ is called a *diagonal* block.

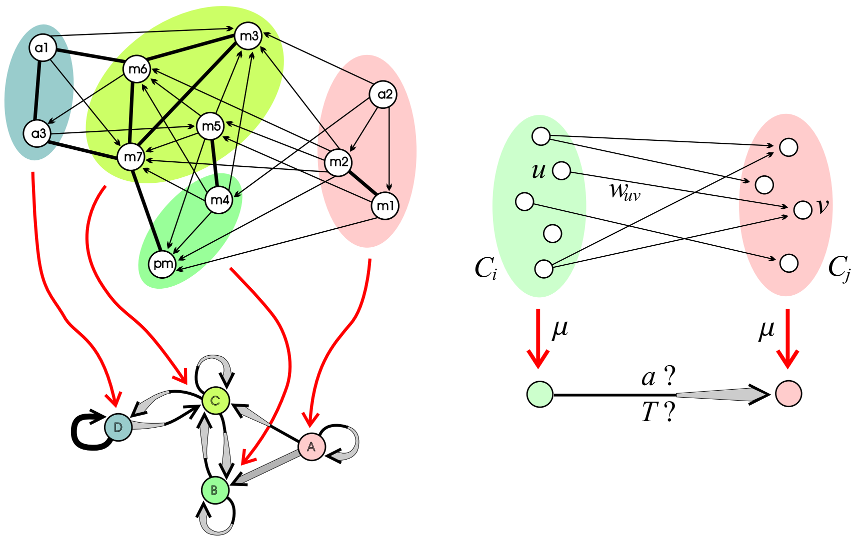


Fig. 11. Blockmodeling

A *blockmodel* consists of structures obtained by identifying all units from the same cluster of the clustering **C**. For an exact definition of a blockmodel we have to be precise also about which blocks produce an arc in the *reduced graph* and which do not, and of what *type*. Some types of connections are presented in Figure 12. The reduced graph can be represented by relational matrix, called also *image matrix*.

Also, by *reordering* of network matrix so that the units from each cluster of the optimal clustering are located together we obtain a matrix representation of the network with visible structure.

How to determine an appropriate blockmodel? The blockmodeling can be formulated as a *clustering problem* (Φ, P) as follows:

Determine the clustering $\mathbf{C}^\star \in \Phi$ for which

$$P(\mathbf{C}^\star) = \min_{\mathbf{C}\in\Phi} P(\mathbf{C})$$

Since the set of units U is finite, the set of feasible clusterings Φ is also finite. Therefore the set $\mathrm{Min}(\Phi, P)$ of all solutions of the problem (optimal clusterings) is not empty. In theory, the set $\mathrm{Min}(\Phi, P)$ can be determined by the complete search – but it turns out that most cases of the clustering problem are $\mathcal{NP}$ hard. The blockmodeling problems are usually solved using local optimization methods based on moving a unit from one cluster to another or interchanging two units between two clusters.

One of the possible ways of constructing a criterion function that directly reflects the considered equivalence is to measure the fit of a clustering to

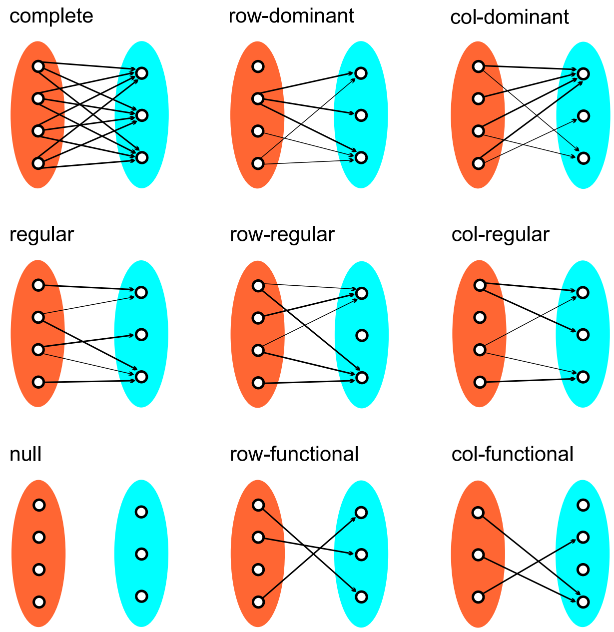


Fig. 12. Block Types

an ideal one with perfect relations within each cluster and between clusters according to the considered equivalence.

Given a clustering $\mathbf{C} = \{C_1, C_2, \ldots, C_k\}$, let $\mathcal{B}(C_u, C_v)$ denote the set of all ideal blocks corresponding to block $R(C_u, C_v)$. Then the global error of clustering $\mathbf{C}$ can be expressed as

$$P(\mathbf{C}) = \sum_{C_u, C_v \in \mathbf{C}} \min_{B \in \mathcal{B}(C_u, C_v)} d(R(C_u, C_v), B)$$

where the term $d(R(C_u, C_v), B)$ measures the difference (error) between the block $R(C_u, C_v)$ and the ideal block B. d is constructed on the basis of characterizations of types of blocks. The function d has to be compatible with the selected type of equivalence. Determining the block error, we also determine the type of the best fitting ideal block (the types are ordered).

The criterion function $P(\mathbf{C})$ is *sensitive* iff $P(\mathbf{C}) = 0 \Leftrightarrow \mathbf{C}$ determines an exact blockmodeling. For all presented block types sensitive criterion functions can be constructed. Once a clustering $\mathbf{C}$ and types of blocks are determined, we can also compute the values of connections by using averaging rules.

In Figure 13 a symmetric acyclic (edge connected inside clusters, acyclic reduced graph) blockmodel [27] of *Student Government* at the University of Ljubljana [35] is presented. The obtained clustering in 4 clusters is almost exact. The only error is produced by the arc $(a3, m5)$.

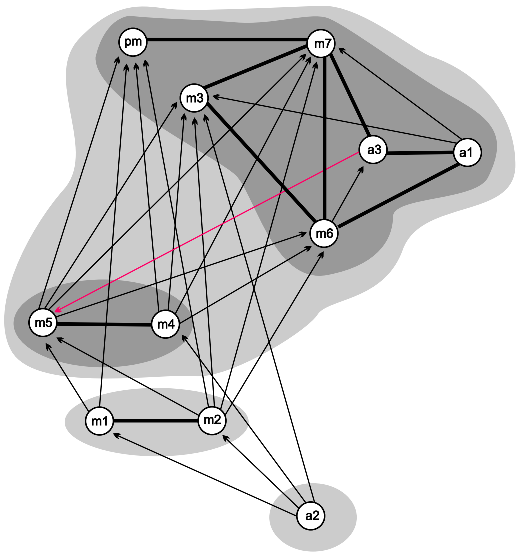


Fig. 13. A Symmetric Acyclic Blockmodel of Student Government

4 Implementation

4.1 Data structures

In `Pajek` analysis and visualization are performed using 6 data types:

- *network* (graph),
- *partition* (nominal or ordinal properties of vertices),
- *vector* (numerical properties of vertices),
- *cluster* (subset of vertices),
- *permutation* (reordering of vertices, ordinal properties), and
- *hierarchy* (general tree structure on vertices).

In the near future we intend to extend this list with a support of multiple networks and partitions of edges.

The power of `Pajek` is based on several transformations that support different transitions among these data structures. Also the menu structure (see Figure 14) of the main `Pajek`'s window is based on them. `Pajek`'s main window uses a 'calculator' paradigm with list-accumulator for each data type. The operations are performed on the currently active (selected) data and are also returning the results through accumulators.

The values of vectors can be used to determine several elements of network display such as: X, Y, Z coordinates and the size of the vertex shape. The partition can be graphically represented by the color and shape of vertices. Also the values of edges can be represented by the thickness and/or color.

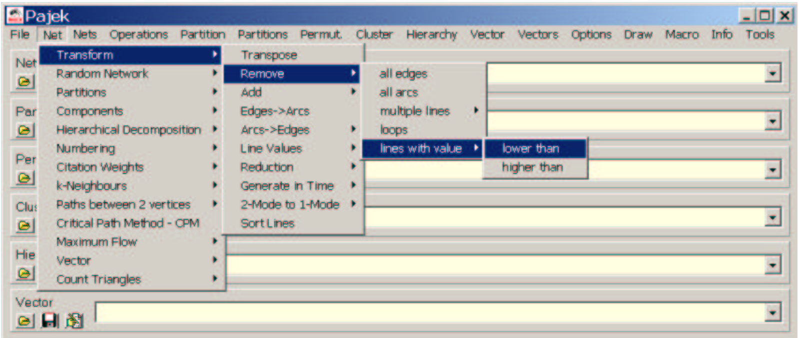


Fig. 14. `Pajek`'s Main Window

4.2 Implemented algorithms

In `Pajek`, besides the algorithms described in section 3, several known efficient algorithms are implemented, like:

- *simplifications and transformations*: deleting loops, multiple edges, transforming arcs to edges etc.;
- *components*: strong, weak, biconnected, symmetric;
- *decompositions*: symmetric-acyclic, hierarchical clustering;
- *paths*: shortest path(s), all paths between two vertices;
- *flows*: maximum flow between two vertices;
- *neighborhood*: k-neighbours;
- *CPM* – critical paths;
- *social networks algorithms*: centrality measures, hubs and authorities, measures of prestige, brokerage roles, structural holes, diffusion partitions;
- *measures of dependencies among partitions / vectors*: Cramer's V, Spearman rank correlation coefficient, Pearson correlation coefficient, Rajski coefficient;
- *extracting* subnetwork;
- *shrinking* clusters in network (generalized blockmodeling);
- *reordering*: topological ordering, Richards's numbering, Murtagh's seriation and clumping algorithms, depth/breadth first search;

`Pajek` contains also some data analysis procedures which have higher order time complexities and can be therefore used only on smaller networks, or selected parts of large networks: hierarchical clustering, generalized blockmodeling, partitioning signed graphs [26], TSP (Traveling Salesman Problem), computing geodesics matrices, etc.

The procedures are available through the main window menus. Frequently used sequences of operations can be defined as *macro*s. This allows also the adaptations of `Pajek` to groups of users from different areas (social networks, chemistry, genealogy, computer science, mathematics. . .) for specific tasks.

4.3 Layout Algorithms and Layout Features

Special emphasis is given in `Pajek` to automatic generation of network layouts. Several standard algorithms for automatic graph drawing are implemented: spring embedders (Kamada-Kawai and Fruchterman-Reingold), layouts determined by eigenvectors (Lanczos algorithm), drawing in layers (genealogies and other acyclic structures), fish-eye views and block (matrix) representation.

These algorithms were modified and extended to enable additional options: drawing with constraints (optimization of the selected part of the network, fixing some vertices to predefined positions, using values of edges as similarities or dissimilarities), drawing in 3D space. `Pajek` also provides tools for manual editing of graph layout.

Properties of vertices/edges (given as data or computed) can be represented using colors, sizes and/or shapes of vertices/edges.

`Pajek` supports also drawing sequences of networks in its Draw window, and exports sequences of networks in suitable formats that can be examined with special 2D or 3D viewers (e.g., SVG and Mage). Pictures in SVG can be further controled using support written in Javascript.

4.4 Interfaces

`Pajek` supports also some non-native input formats: UCINET DL files [53]; Vega graph files [54]; chemical MDLMOL [41] and BS; and genealogical GEDCOM [30].

The layouts can be exported in the following output graphic formats that can be examined by special 2D and 3D viewers: Encapsulated PostScript (EPS) [31], Scalable Vector Graphics (SVG) [1], VRML [24], MDLMOL/chime [41], and Kinemages (Mage) [49].

The main window menu *Tools* provides export of `Pajek`'s data to statistical program R [48,21]. In the *Tools* menu, the user can prepare calls to her/his favorite viewers and other tools. It is also possible to run `Pajek` (+macros) from other programs (R, Ucinet, and others).

5 Examples

Several examples of applications of `Pajek` were already presented as illustrations while describing selected algorithms.

In Figure 15 a 3D layout of a graph obtained using *eigenvector*s is presented.

In Figure 16 a snapshoot of 3D layout displayed in a VRML viewer of our drawing of graph A from the Graph drawing contest 1997 is presented [33].

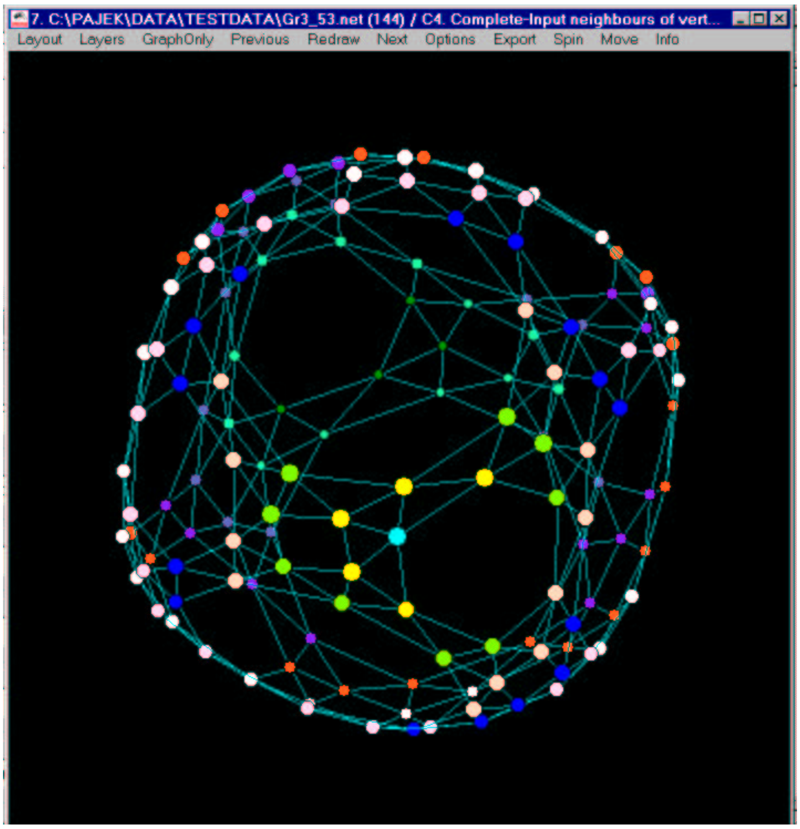


Fig. 15. 3D layout obtained using eigenvectors

6 Software

6.1 Architecture

`Pajek` is implemented in Delphi and runs on Windows operating systems. On the *things to do* list we have: support for GraphML format, implementing `Pajek` on Unix, and replacing macros by a Javascript(?) based network scripting language.

6.2 Availability

`Pajek` is still under development. The latest version is freely available, for noncommercial use, at its home page:
`http://vlado.fmf.uni-lj.si/pub/networks/pajek/`

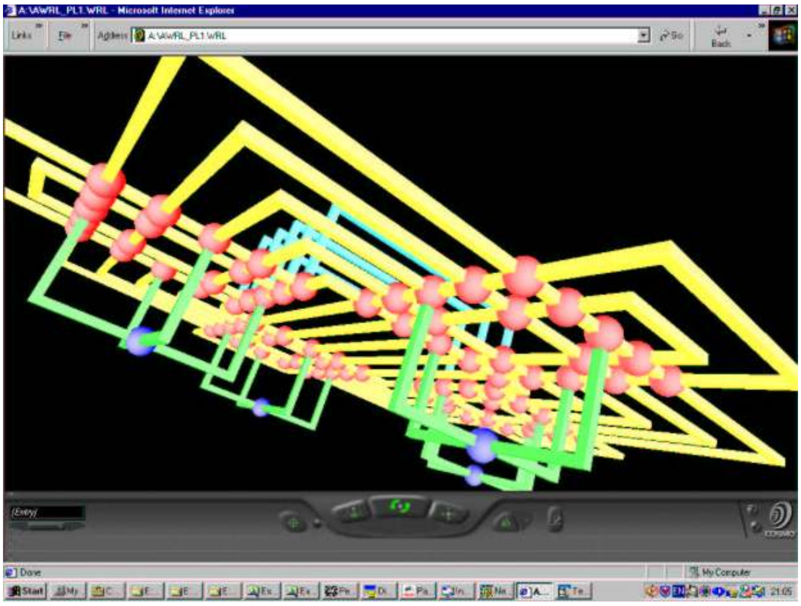


Fig. 16. GD'97 contest graph A in VRML

References

1. Adobe SVG Viewer (2002) `http://www.adobe.com/svg/viewer/install/`
2. Batagelj V. (1986) Graph – data structure and algorithms in pascal. Research report.
3. Batagelj, V. (1997) Notes on blockmodeling. *Social Networks* **19**, 143-155.
4. Batagelj V. (2002) Efficient Algorithms for Citation Network Analysis
5. Batagelj V., Brandes U. (2002) *Fast generation of large sparse random graphs.* in preparation.
6. Batagelj, V., Doreian, P., and Ferligoj, A. (1992) An Optimizational Approach to Regular Equivalence. *Social Networks* **14**, 121-135.
7. Batagelj V., Ferligoj A. (2000) Clustering relational data. Data Analysis (ed.: W. Gaul, O. Opitz, M. Schader), Springer, Berlin, 3-15.
8. Batagelj V., Mrvar A. (1995) Towards NetML Networks Markup Language. Presented at International Social Network Conference, London, July 6-10, 1995. `http://www.ijp.si/ftp/pub/preprints/ps/95/trp9515.ps`
9. Batagelj V., Mrvar A. (1991-94) Programs for Network Analysis. `http://vlado.fmf.uni-lj.si/pub/networks/`
10. Batagelj V., Mrvar A. (1998) `Pajek` – A Program for Large Network Analysis. Connections, **21 (2)**, 47-57
11. Batagelj V., Mrvar A. (2000) Some Analyses of Erdős Collaboration Graph. Social Networks, **22**, 173-186
12. Batagelj V., Mrvar A. (2001) A Subquadratic Triad Census Algorithm for Large Sparse Networks with Small Maximum Degree. Social Networks, **23**, 237-243

13. Batagelj V., Mrvar A. (2002) `Pajek` - Analysis and Visualization of Large Networks. In: Mutzel P., Jünger M., Leipert S. (Eds.) GD'01, Vienna, Austria. September 23-26, 2001 LNCS **2265**. Springer-Verlag, 477-478.
14. Batagelj V., Mrvar A. (2002) Density based approaches to Reuters terror news network analysis. submitted.
15. Batagelj V., Pisanski T. (1989) Xgraph project documentation.
16. 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.
17. Batagelj V., Mrvar A., Zaveršnik M. (2002) Network analysis of texts. Language Technologies, Ljubljana, p. 143-148.
18. Batagelj V., Zaveršnik M. (2001) An $O(m)$ Algorithm for Cores Decomposition of Networks. Submitted.
19. Batagelj V., Zaveršnik M. (2002) Generalized Cores. Submitted. `http://arxiv.org/abs/cs.DS/0202039`
20. Batagelj, V. and Zaveršnik, M. (2002) Triangular connectivity and its generalizations, in preparation.
21. Butts, C.T. (2002) sna: Tools for Social Network Analysis. `http://cran.at.r-project.org/src/contrib/PACKAGES.html#sna`
22. Caida: Internet Visualization Tool Taxonomy. `http://www.caida.org/tools/taxonomy/visualization/`
23. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C. (2001) Introduction to Algorithms, Second Edition. MIT Press.
24. Cosmo Player (2002) `http://ca.com/cosmo/`
25. de Nooy W., Mrvar A., Batagelj V. (2002) *Exploratory Social Network Analysis With* `Pajek`. to be published by the Cambridge University Press.
26. Doreian P., Mrvar A. (1996) A Partitioning Approach to Structural Balance. Social Networks, **18**. 149-168
27. Doreian, P., Batagelj, V., Ferligoj, A. (2000) Symmetric-acyclic decompositions of networks. *J. classif.*, **17**(1), 3-28.
28. 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).
29. Garfield E, Sher IH, and Torpie RJ.: The Use of Citation Data in Writing the History of Science. Philadelphia: The Institute for Scientific Information, December 1964. `http://www.garfield.library.upenn.edu/papers/useofcitdatawritinghistofsci.pdf`
30. GEDCOM 5.5. `http://homepages.rootsweb.com/~pmcbride/gedcom/55gctoc.htm`
31. Ghostscript, Ghostview and GSview `http://www.cs.wisc.edu/~ghost/`
32. Gibbons A. (1985) Algorithmic Graph Theory. Cambridge University Press.
33. Graph Drawing Contest 1997. `http://vlado.fmf.uni-lj.si/pub/gd/gd97.htm`
34. Grossman J. (2002) The Erdős Number Project. `http://www.oakland.edu/~grossman/erdoshp.html`
35. Hlebec, V. (1993) Recall versus recognition: Comparison of two alternative procedures for collecting social network data. Metodološki zvezki **9**, Ljubljana: FDV, 121-128.
36. Hummon, N.P. & Doreian, P. (1989) Connectivity in a citation network: The development of DNA theory. *Social Networks*, **11**, 39–63.

37. Jones B. (2002). Computational geometry database. `http://compgeom.cs.uiuc.edu/~jeffe/compgeom/biblios.html`
38. 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` `http://citeseer.nj.nec.com/kleinberg97authoritative.html`
39. Knuth, D. E. (1993) *The Stanford GraphBase*. Stanford University, ACM Press, New York. `ftp://labrea.stanford.edu/pub/sgb/`
40. Mahnken, I. (1960) Dubrovački patricijat u XIV veku. Beograd, Naučno delo.
41. MDL Information Systems, Inc. (2002) `http://www.mdli.com/`
42. James Moody home page (2002) `http://www.soc.sbs.ohio-state.edu/jwm/`
43. Mrvar A., Batagelj V. (2000) Relational Calculator - a tool for analyzing social networks. Metodološki zvezki 16, FDV, Ljubljana, 63-76.
44. Murtagh, F. (1985) Multidimensional Clustering Algorithms, *Compstat lectures*, **4**, Vienna: Physica-Verlag.
45. ODLIS (2002) Online dictionary of library and information science. `http://vax.wcsu.edu/library/odlis.html`
46. Pajek's datasets. `http://vlado.fmf.uni-lj.si/pub/networks/data/`
47. D.M. Pennock etal. (2002) Winners dont't take all, PNAS, **99**/8, 5207-5211.
48. The R Project for Statistical Computing. `http://www.r-project.org/`
49. Richardson D.C., Richardson J.S. (2002) The Mage Page. `http://kinemage.biochem.duke.edu/index.html`
50. Scott, J. (2000) Social Network Analysis: A Handbook, 2nd edition. London: Sage Publications.
51. Seidman S. B. (1983) Network structure and minimum degree, *Social Networks*, **5**, 269–287.
52. Tarjan, R. E. (1983) *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics Philadelphia, Pennsylvania.
53. UCINET (2002) `http://www.analytictech.com/`
54. Project Vega (2002) `http://vega.ijp.si/`
55. Wasserman S., Faust K. (1994) *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge.
56. Wasserman, S., and Pattison, P. (1996) Logit models and logistic regressions for social networks: I. An introduction to Markov graphs and p^*. Psychometrika, 60, 401-426. `http://kentucky.psych.uiuc.edu/pstar/index.html`
57. White D.R., Batagelj V., Mrvar A. (1999) Analyzing Large Kinship and Marriage Networks with Pgraph and Pajek. Social Science Computer Review, **17 (3)**, 245-274
58. Wilson, R.J., Watkins, J.J. (1990) *Graphs: An Introductory Approach*. New York: John Wiley and Sons.
59. Yuen Ho, et.al. (2002) Systematic identification of protein complexes in Saccharomyces cerevisiae by mass spectrometry. Nature, vol 415, 180-183. `http://www.mshri.on.ca/tyers/pdfs/proteome.pdf`
60. Zaveršnik M., Batagelj V., Mrvar A. (2002) Analysis and visualization of 2-mode networks. Proceedings of Sixth Austrian, Hungarian, Italian and Slovenian Meeting of Young Statisticians, October 5-7, 2001, Ossiach, Austria. University of Klagenfurt, p. 113-123.