Some Approaches to the Analysis of Large Networks

in preparation

Vladimir Batagelj
University of Ljubljana

Based on slides for:

Annual Meeting of Research Cluster 1126: ***Algorithmic Aspects of Large and Complex Networks***, University of Konstanz, July 22-24, 2002

Student conference ***Knowledge Extraction***, University of Ljubljana, FMP, August 18-25, 2002

Outline

- Pajek
- Genealogies and p-graphs (with Andrej Mrvar)
- Cores and p-cores (with Matjaž Zaveršnik)
- Citation networks
- Reuters news after September 11th (with Andrej Mrvar)
- Connectivity by short cycles (with Matjaž Zaveršnik)

Pajek

`Pajek` (Slovene word for Spider) is a program, for Windows (32 bit), for analysis of *large networks*. It is freely available, for noncommercial use, at its homepage: **`http://vlado.fmf.uni-lj.si/pub/networks/pajek/`**

With Andrej Mrvar, we started to develop **`Pajek`** in November 1996. Some procedures were contributed by Matjaž Zaveršnik.

Large Networks

Large networks can be found in many different areas. Usually they are produced automatically, using computers, from different data sources that are already available in computer readable form. For example:

- transportation and communication networks;
- flow graphs of programs;
- large molecule;
- large genealogies;
- networks derived from dictionaries and other texts;
- bibliographies, citation networks, . . .

Such networks cannot be treated efficiently using standard network analysis tools.

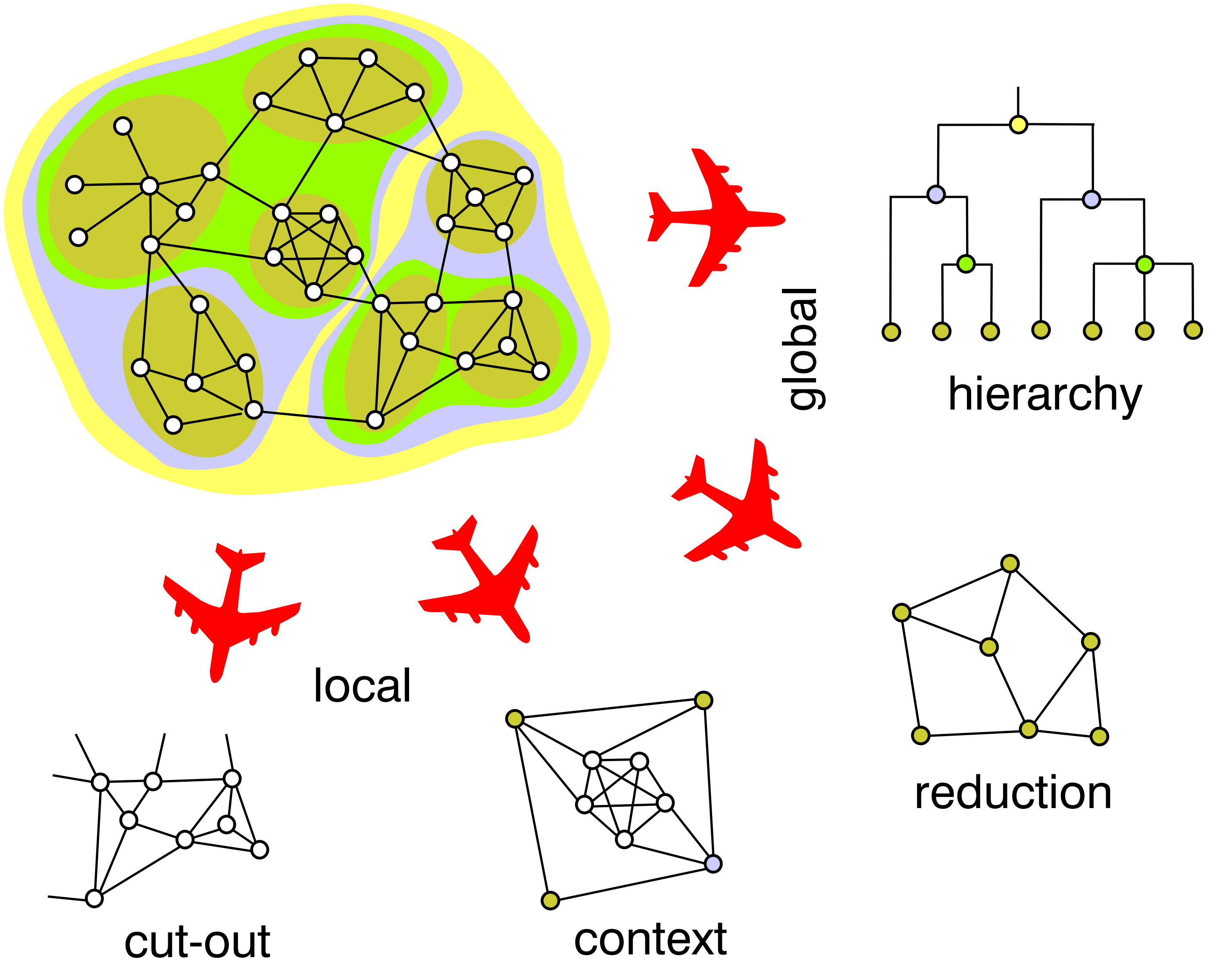
Pajek – Goals

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

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

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

Pajek – Goals



Pajek Data Structures

Currently **`Pajek`** uses six data structures to implement the algorithms:

- *network* – main object (vertices and lines);
- *partition* – to which cluster a vertex belongs;
- *vector* – values of vertices;
- *permutation* – reordering of vertices;
- *cluster* – subset of vertices (e.g. a cluster from partition);
- *hierarchy* – hierarchically ordered clusters and vertices.

The power of **`Pajek`** is based on several transformations which support different transitions among these data structures.

Besides its own input formats, **`Pajek`** supports several other formats: `UCINET DL`, genealogical `GED`, and some molecular formats: `BS` (Ball and Stick), `MAC` (Mac Molecule) and `MOL` (MDL MOLfile).

Algorithms

Besides standard algorithms for analysis of large networks (connectivities, topological sort, shortest paths, ...) **Pajek** contains also some new algorithms:

- *pattern search* – appearances of selected small network inside the large;
- *triadic census* – distribution of triads;
- *centrality indices* – Brandes's algorithms;
- *hubs and authorities* – Kleinberg's algorithm;
- *p-graphs* – representation of genealogies and their analysis;
- *cores* and *p-cores* – identification of dense parts of network;
- *citation weights* – Hummon and Doreian's methods for citation network analysis;
- *short cycle connectivity* – elaboration of Everett's cyclic decomposition of networks (EBLOC);
- *generalized blockmodeling*.

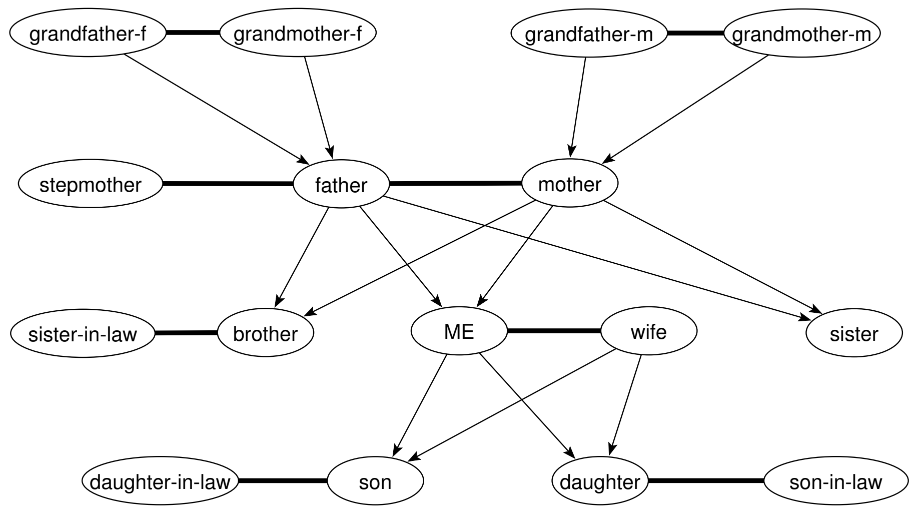
Demo

```
Network/Read [type=bs]/DNA.bs
Net/Partitions/Vertex Labels
Draw/Draw-Partition
draw:
Options/Transform/Fit Area/max(x),...
[move  X,Y,Z]
main:
Network/Read [type=net]/C5.net
Nets/First Network
[select network DNA.bs]
Nets/Second Network
Nets/Fragment (1 in 2)/Find [opts: induced,extract]
Partitions/Second Partition
[select partition Name partition of N1]
Partitions/First Partition
Partitions/Extract Second from First [1 1]
Draw/Draw-Partition
```

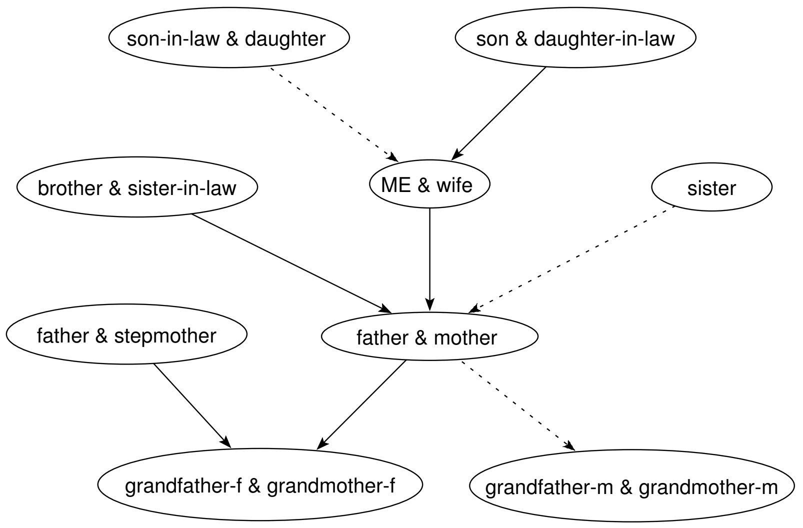
Genealogies

GEDCOM standard `http://www.gendex.com/gedcom55/55gctoc.htm`

Ore graph



p-graph



The p-graph representation provides simpler picture of genealogy and is also easier to analyse – acyclic graph.

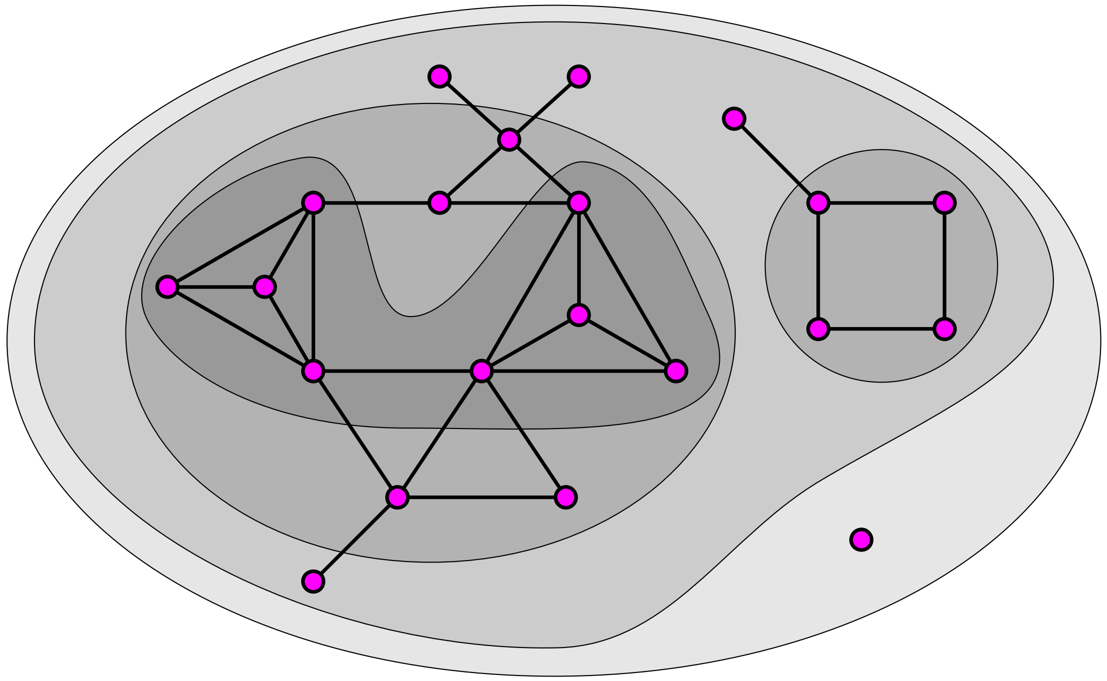
Cores

The notion of core was introduced by Seidman in 1983.

Let $\mathbf{G} = (V, L)$ be a simple graph. V is the set of *vertices* and L is the set of *lines* (*edges* or *arcs*). We will denote $n = |V|$ and $m = |L|$. A subgraph $\mathbf{H} = (C, L|C)$ induced by the set C is a *k-core* or a *core of order k* iff $\forall v \in C : \deg_H(v) \geq k$ and $\mathbf{H}$ is a maximum subgraph with this property. The core of maximum order is also called the *main core*. The *core number* of vertex v is the highest order of a core that contains this vertex. Since the set C determines the corresponding core $\mathbf{H}$ we also often call it a core.

The degree $\deg(v)$ can be the number of neighbors in an undirected graph or in-degree, out-degree, in-degree + out-degree, $\ldots$ determining different types of cores.

Properties of the cores



- The cores are nested: $i < j \implies \mathbf{H}_j \subseteq \mathbf{H}_i$
- Cores are not necessarily connected subgraphs.
- There exists very efficient algorithm to determine core numbers.

Algorithm for cores

If from a given graph $G = (V, L)$ we recursively delete all vertices, and lines incident with them, of degree less than k, the remaining graph is the k-core. It can be implemented in $O(m)$.

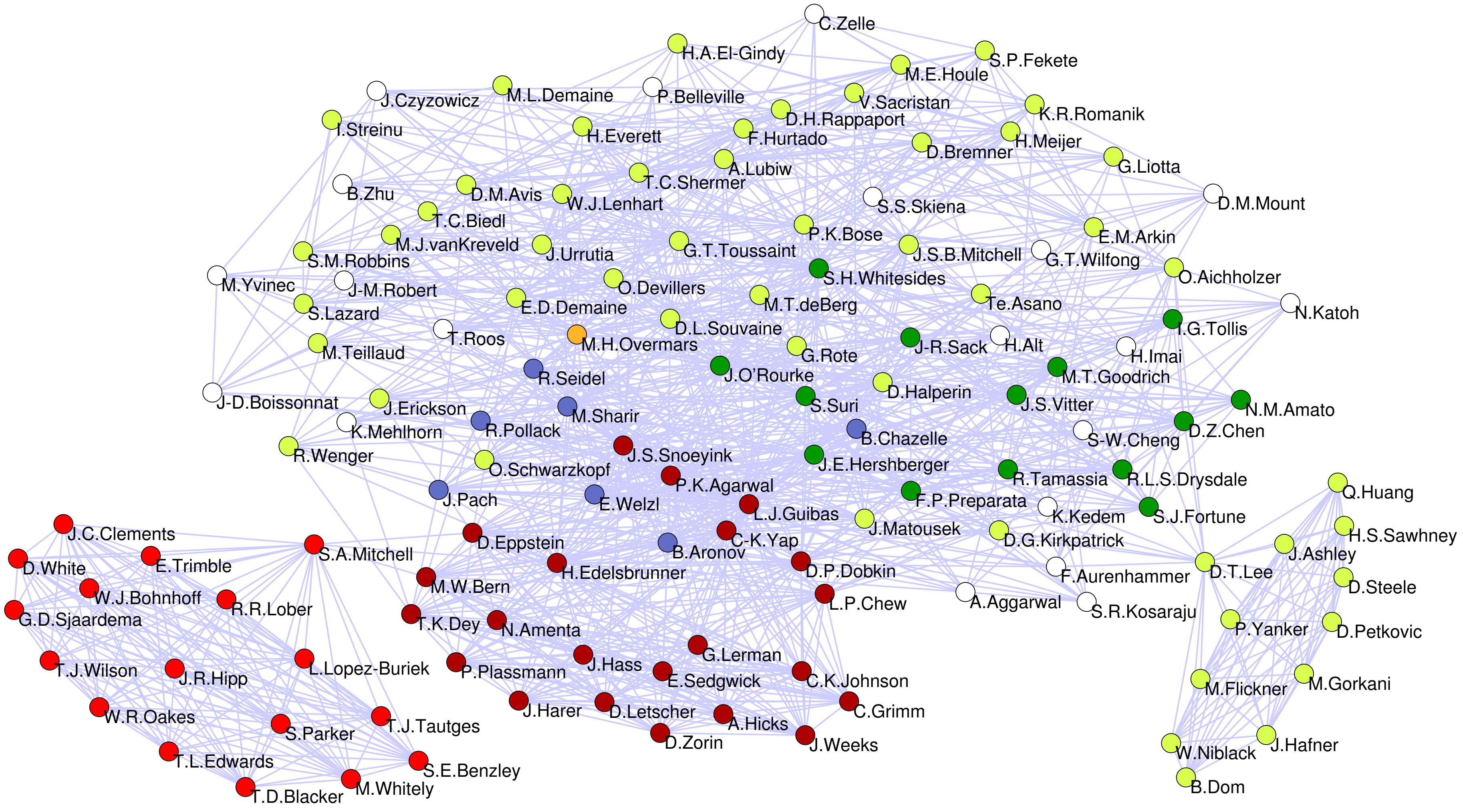
```
INPUT: graph $G = (V, L)$ represented by lists of neighbours
OUTPUT: table $core$ with core number for each vertex
1.1      compute the degrees of vertices;
1.2      order the set of vertices $V$ in increasing order of their degrees;
2        for each $v \in V$ in the order do
2.1          $core[v] := degree[v]$;
2.2          for each $u \in Neighbours(v)$ do
2.2.1            if $degree[u] > degree[v]$ then
2.2.1.1              $degree[u] := degree[u] - 1$;
2.2.1.2              reorder $V$ accordingly
```

Example: Computational Geometry

We applied the described algorithm for cores on the ***authors collaboration network*** based on the BibTEX bibliography obtained from the ***Computational Geometry Database*** `geombib`. Two authors are linked with an edge, iff they wrote a common paper. Using a simple program in Python, the BibTEX data were transformed into the corresponding network, and output to the file in Pajek format. The obtained network has 9072 vertices (authors) and 22577 edges / 13567 edges as a simple network.

The problem with the obtained network is that it contains several vertices corresponding to the same author (Pankaj K. Agarwal, P. Agarwal, Pankaj Agarwal, and P.K. Agarwal) – that are easy to guess; but an ’insider’ information is needed to know that O. Schwarzkopf and O. Cheong are the same person. We manually produced the ***name equivalence partition*** and then shrank the network according to it. The reduced simple network contains 7343 vertices and 11898 edges.

Computational geometry 10 core



Demo

```
Read network Geom.net
Net / Partitions / Core / Input
Info / Partition
Operations / Extract from Network / Partition [10,99]
Net / Partitions / Core / Input
Draw / Draw-Partition
Layout / Energy / Kamada-Kawai / Free
Operations / Shrink Network / Partition [1,21]
Draw / Draw-Partition
```

Generalized cores

Vertex property functions

Let $\mathbf{N} = (V, L, w)$ be a ***network***, where $\mathbf{G} = (V, L)$ is a graph and $w : L \to \mathbb{R}$ is a function assigning values to lines.

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.

Examples of vertex property functions

Let $N(v)$ denotes the set of neighbors of vertex v in graph G, and $N(v, U) = N(v) \cap U, U \subseteq V$.

1. $p_1(v, U) = \deg_U(v)$
2. $p_2(v, U) = \text{indeg}_U(v)$
3. $p_3(v, U) = \text{outdeg}_U(v)$
4. $p_4(v, U) = \text{indeg}_U(v) + \text{outdeg}_U(v)$
5. $p_5(v, U) = \sum_{u \in N(v,U)} w(v, u)$, where $w : L \rightarrow \mathbb{R}_0^+$
6. $p_6(v, U) = \max_{u \in N(v,U)} w(v, u)$, where $w : L \rightarrow \mathbb{R}$
7. $p_7(v, U) =$ number of cycles of length k through vertex v

p-cores

The subgraph $\mathbf{H} = (C, L|C)$ induced by the set $C \subseteq V$ is a ***p-core at level t*** $\in \mathbb{R}$ iff

- $\forall v \in C : t \leq p(v, C)$
- C is maximal such set.

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

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

All among functions $p_1 - p_7$ are monotone.

Monotone p functions and cores

For monotone function the p-core at level t can be determined by successively deleting vertices with value of p lower than t.

```
$C := V$;
**while** $\exists v \in C : p(v, C) < t$ **do** $C := C \setminus \{v\}$;
```

Theorem 1 *For monotone function p the above procedure determines the p-core at level t.*

Corolary 1 *For monotone function p the cores are nested*

$$t_1 < t_2 \Rightarrow \mathbf{H}_{t_2} \subseteq \mathbf{H}_{t_1}$$

Example of nonmonotone p function

The p function

$$p(v,U) = \begin{cases} 0 & N(v,U) = \emptyset \\ \dfrac{1}{|N(v,U)|} \displaystyle\sum_{u \in N(v,U)} w(v,u) & \text{otherwise} \end{cases}$$

where $w : L \to \mathbb{R}_0^+$, is not monotone.

Local functions

The p function is ***local*** iff

$$p(v, U) = p(v, N(v, U))$$

The functions $p_1 - p_6$ from examples are local; p_7 is **not** local for $k \geq 4$.

In the following we shall assume also that for the function p there exists a constant p_0 such that

$$\forall v \in V : p(v, \emptyset) = p_0$$

For a local p function an $O(m \max(\Delta, \log n))$ algorithm for determining p-core at level t exists (assuming that $p(v, N(v, C))$ can be computed in $O(\deg_C(v))$).

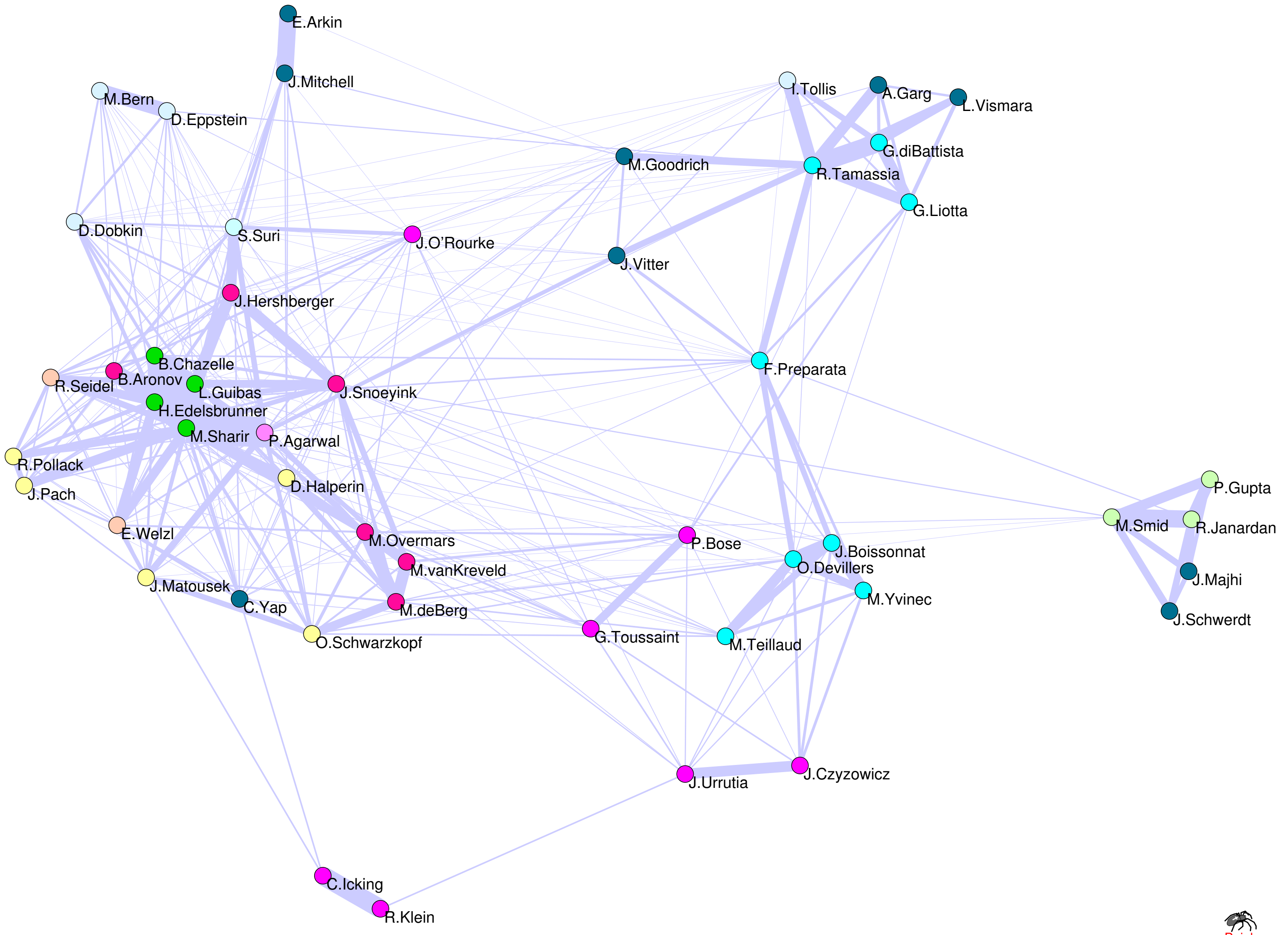
Determining the hierarchy of p-cores

The hierarchy is determined by the core-number assigned to each vertex – the highest level value of p-cores that contain the vertex.

INPUT: graph $G = (V, L)$ represented by lists of neighbors
OUTPUT: table *core* with core number for each vertex

```
1.       $C := V$;
2.       **for** $v \in V$ **do** $p[v] := p(v, N(v, C))$;
3.       $build\_min\_heap(v, p)$;
4.       **while** $sizeof(heap) > 0$ **do begin**
4.1.         $C := C \setminus \{top\}$;
4.2.         $core[top] := p[top]$;
4.3.         **for** $v \in N(top, C)$ **do begin**
4.3.1.           $p[v] := \max\{p[top], p(v, N(v, C))\}$;
4.3.2.           $update\_heap(v, p)$;
             **end**;
         **end**;
```

Computational geometry value core 46



Paiek

Example – Internet Connections

As an example of application of the proposed algorithm we applied it to the routing data on the Internet network. This network was produced from web scanning data (May 1999) available from

`http://www.cs.bell-labs.com/who/ches/map/index.html`
It can be obtained also as a **`Pajek`**'s NET file from

`http://vlado.fmf.uni-lj.si/`
`pub/networks/data/web/web.zip`
It has 124 651 vertices, 195 029 arcs (loops were removed), $\Delta = 151$, and average degree is 3.13. The arcs have as values the number of *traceroute paths* which contain the arc.

Using **`Pajek`** implementation of the proposed algorithm on 300 MHz PC we obtained in 3 seconds the p_5-cores segmentation presented in the table – there are n_k vertices with p_5-core number in the interval $(t_{k-1}, t_k]$.

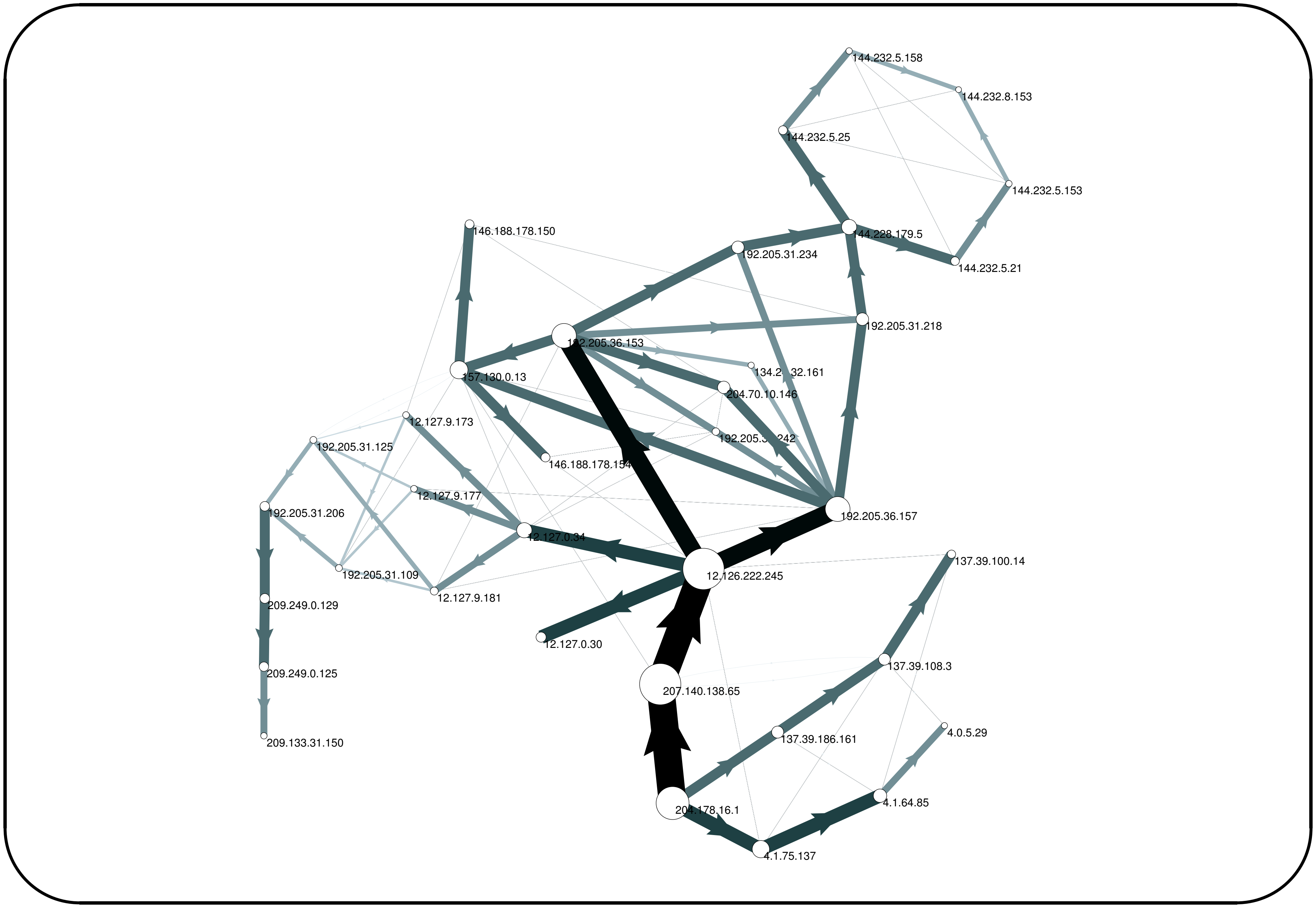
... Example – Internet Connections

Table 1: p_5-cores of the Routing Data Network at Different Levels.

k	t	n	k	t	n
1	1	7582	12	2048	490
2	2	9288	13	4096	314
3	4	12519	14	8192	153
4	8	33866	15	16384	48
5	16	33757	16	32768	44
6	32	11433	17	65536	11
7	64	6518	18	131072	9
8	128	3812	19	262144	0
9	256	2356	20	524288	2
10	512	1528	21	1048576	3
11	1024	918			

... Example – Internet Connections

The program also determined the p_5-core number for every vertex. The figure shows a p_5-core at level 25000 of the Internet network – every vertex inside this core is visited by at least 25000 traceroute paths. In the figure the sizes of circles representing vertices are proportional to (the square roots of) their p_5-core numbers. Since the arcs values span from 1 to 626826 they can not be displayed directly. We recoded them according to the thresholds $1000 \cdot 2^{k-1}$, $k = 1, 2, 3 \ldots$. These class numbers are represented by the thickness of the arcs.


144.232.5.158
144.232.8.153
144.232.5.25
144.232.5.153
144.228.179.5
144.232.5.21
146.188.178.150
192.205.31.234
192.205.31.218
192.205.36.153
157.130.0.13
134.2 32.161
204.70.10.146
12.127.9.173
192.205.3 .242
192.205.31.125
146.188.178.154
12.127.9.177
192.205.36.157
192.205.31.206
12.127.0.34
137.39.100.14
12.126.222.245
192.205.31.109
12.127.9.181
209.249.0.129
12.127.0.30
209.249.0.125
137.39.108.3
207.140.138.65
4.0.5.29
209.133.31.150
137.39.186.161
4.1.64.85
204.178.16.1
4.1.75.137