Cores Decomposition of Networks

Vladimir Batagelj, Matjaž Zaveršnik
University of Ljubljana, Slovenia

Recent Trends in Graph Theory,
Algebraic Combinatorics, and Graph Algorithms

September 24–27, 2001, Bled, Slovenia

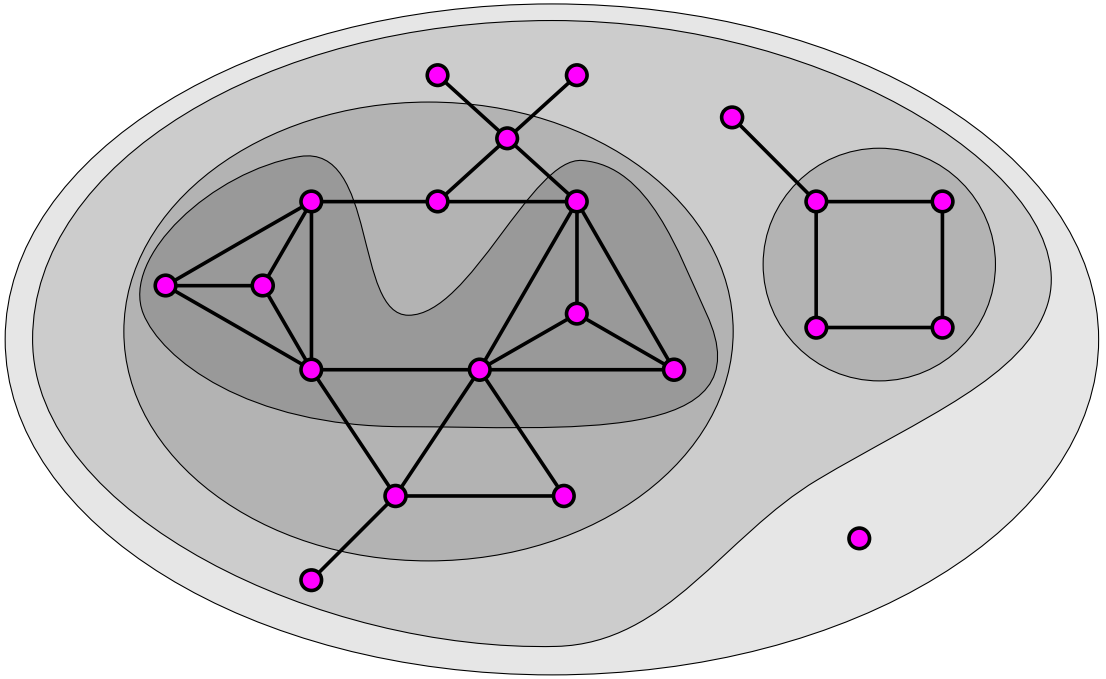
Cores

The notion of core was introduced by Seidman in 1983.

Let $G = (V, L)$ 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 $H = (W, L|C)$ 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 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 smallest order of a core that contains this vertex.

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

Properties of the cores



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

p-cores

Let $G = (V, L)$ be a graph and $p(v, U)$, $v \in V$, $U \subseteq V$ a function with real values.

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

Monotone p functions and cores

The function $p(v, U)$ is *monotone* iff it holds

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

We assume also that $\forall v \in V : p(v, \emptyset) = const$.

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 returns the p-core at level t.*

Proof

The set C returned by the procedure evidently has the first property from the core definition. Let us also show that for monotone p the result of the procedure is independent of the order of deletions.

Suppose the contrary – there are two different p-cores at level t, denoted C and D. The core C was produced by deleting the sequence $u_1, u_2, u_3, \ldots, u_p$; and D by the sequence $v_1, v_2, v_3, \ldots, v_q$. Assume that $D \setminus C \neq \emptyset$. We show that this leads to contradiction.

Take any $z \in D \setminus C$. We shall show that it also can be deleted. To see this, we first apply the sequence $v_1, v_2, v_3, \ldots, v_q$ to get D. Since $z \in D \setminus C$ it appears in sequence $u_1, u_2, u_3, \ldots, u_s = z$. Let $U_0 = \emptyset$ and $U_i = U_{i-1} \cup \{u_i\}$. Then, since $\forall i \in 1..p : p(u_i, V \setminus U_{i-1}) < t$, we have, by monotonicity of p, also $\forall i \in 1..p : p(u_i, (V \setminus D) \setminus U_{i-1}) < t$. Therefore also all $u_i \in D \setminus C$ are deleted – $D \setminus C = \emptyset$ – a contradiction.

Since the result of the procedure is uniquely defined and vertices outside C have p value lower than t, the final set C satisfies also the second condition from the definition of p-core – it is the p-core.

Examples of monotone p functions

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

1. $p_1(v,U) = \deg_U(v)$
2. $p_2(v,U) = in\,\deg_U(v)$
3. $p_3(v,U) = out\,\deg_U(v)$
4. $p_4(v,U) = in\,\deg_U(v) + out\,\deg_U(v)$
5. $p_5(v,U) = \dfrac{\deg_U(v)}{\deg(v)}, \quad \deg(v) > 0; \quad 0$ otherwise
6. $p_6(v,U) = \sum_{u \in N(v,U)} a(v,u)$, where $a : L \to \mathbb{R}_0^+$
7. $p_7(v,U) = \max_{u \in N(v,U)} a(v,u)$, where $a : L \to \mathbb{R}$

Example of nonmonotone p function

$$p(v,U) = \frac{1}{|N(v,U)|} \sum_{u \in N(v,U)} a(v,u)$$

where $a : L \to \mathbb{R}_0^+$. $p(v,U) = 0$ if $N(v,U) = \emptyset$.

In the graph $G = (V,L)$, $V = \{a,b,c,d,e,f\}$,

L	$(a:b)$	$(b:c)$	$(c:d)$	$(e:f)$	$(f:c)$
a	3	1	4	3	1

we get different results depending on whether we first delete the vertex c or b (or f).

Local function

The p function is *local* iff

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

For local p function an $O(m \max(\Delta, \log n))$ algorithm exists (assuming that $p(v, N(v, C))$ can be computed in $O(\deg_C(v))$).

Algorithm

INPUT: graph $G = (V, L)$ represented by lists of neighbors and $t \in \mathbb{R}$
OUTPUT: $C \subseteq V$, C is a p-core at level t

1. $C := V$;
2. **for** $v \in V$ **do** $p[v] := p(v, N(v, C))$;
3. $buildminheap(v, p)$;
4. **while** $p[top] < t$ **do begin**
4.1. $C := C \setminus \{top\}$;
4.2. **for** $v \in N(top, C)$ **do begin**
4.2.1. $p[v] := p(v, N(v, C))$;
4.2.2. $updateheap(v, p)$;
end;
end;

The step 4.2.1. can often be speeded up by updating the $p[v]$.

Determining the hierarchy of p-cores

INPUT: graph $G = (V, L)$ represented by lists of neighbors

OUTPUT: table *core* with core number for each vertex

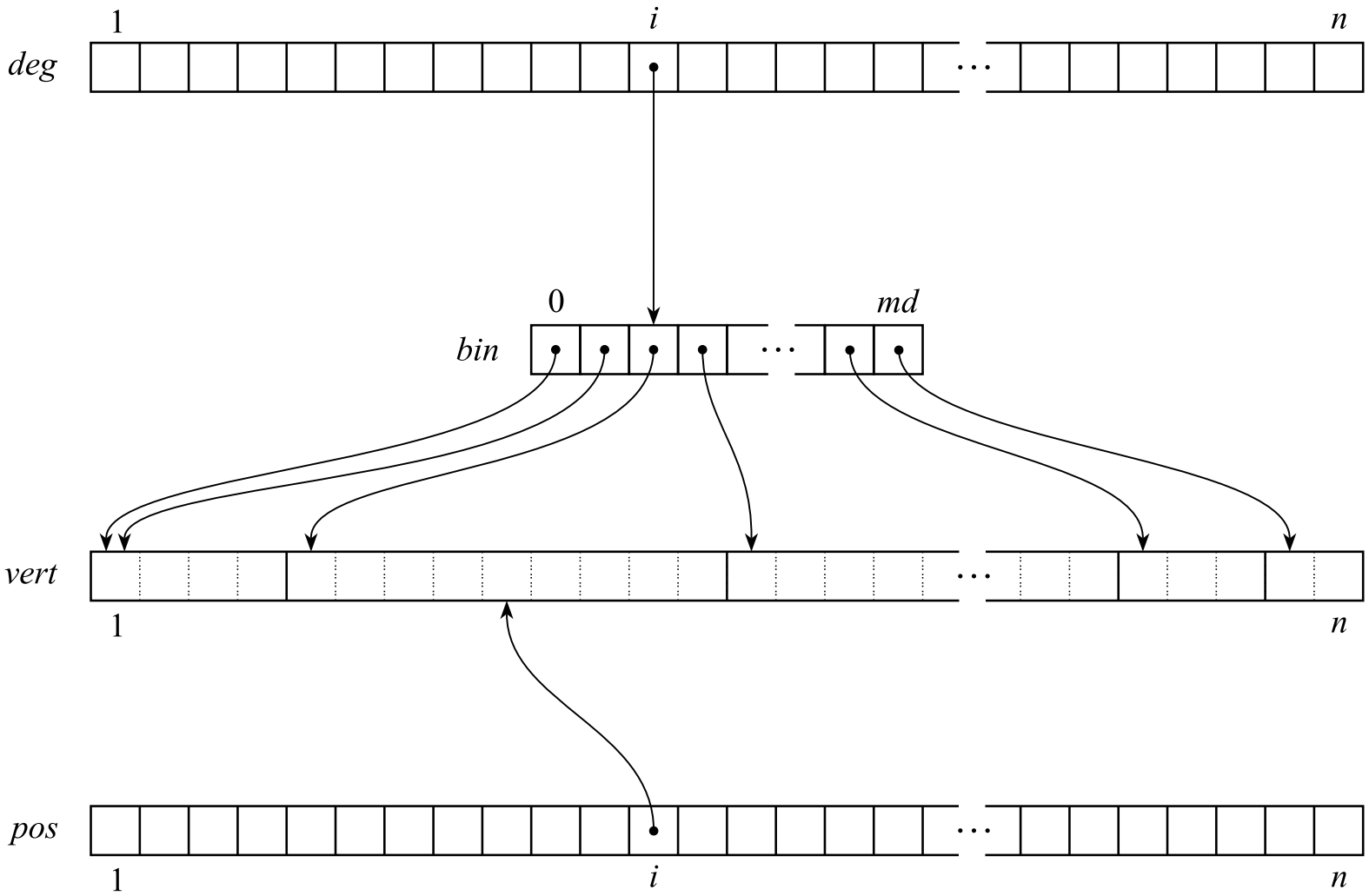
1. $C := V$;
2. **for** $v \in V$ **do** $p[v] := p(v, N(v, C))$;
3. $buildminheap(v, p)$;
4. **while** sizeof$(heap) > 0$ **do begin**
4.1. $C := C \setminus \{top\}$;
4.1 $core[top] := p[top]$;
4.2. **for** $v \in N(top, C)$ **do begin**
4.2.1. $p[v] := p(v, N(v, C))$;
4.2.2. $updateheap(v, p)$;
end;
end;

An $O(m)$ algorithm for degree-cores

INPUT: graph $G = (V, L)$ represented by lists of neighbors
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 begin**
2.1 $core[v] := degree[v]$;
2.2 **for each** $u \in Neighbors(v)$ **do**
2.2.1 **if** $degree[u] > degree[v]$ **then begin**
2.2.1.1 $degree[u] := degree[u] - 1$;
2.2.1.2 reorder V accordingly
end
end;

Data structure



Implementation

```
01 procedure cores(var g: graph; var deg: tableVert);
02 var
03   n, d, md, i, start, num: integer;
04   v, u, w, du, pu, pw: integer;
05   vert, pos: tableVert;
06   bin: tableDeg;
07 begin
08   n := size(g);  md := 0;
09   for v := 1 to n do begin
10     d := 0;  for u in Neighbors(g, v) do inc(d);
11     deg[v] := d;  if d > md then md := d;
12   end;
13   for d := 0 to md do bin[d] := 0;
14   for v := 1 to n do inc(bin[deg[v]]);
15   start := 1;
16   for d := 0 to md do begin
17     num := bin[d];  bin[d] := start;
18     inc(start, num);
19   end;
```

```
20    for v := 1 to n do begin
21      pos[v] := bin[deg[v]];  vert[pos[v]] := v;
22      inc(bin[deg[v]]);
23    end;
24    for d := md downto 1 do bin[d] := bin[d-1];
25    bin[0] := 1;
26    for i := 1 to n do begin
27      v := vert[i];
28      for u in Neighbors(g, v) do begin
29        if deg[u] > deg[v] then begin
30          du := deg[u];   pu := pos[u];
31          pw := bin[du];  w := vert[pw];
32          if u <> w then begin
33            pos[u] := pw;  vert[pu] := w;
34            pos[w] := pu;  vert[pw] := u;
35          end;
36          inc(bin[du]);  dec(deg[u]);
37        end;
38      end;
39    end;
40  end;
```

Time complexity

We shall show that the described algorithm runs in time $O(\max(m, n))$.

To compute (08-12) the degrees of all vertices we need time $O(\max(m, n))$ since we have to consider each line at most twice. The *bin sort* (13-25) consists of five loops of size at most n with constant time $O(1)$ bodies – therefore it runs in time $O(n)$.

The statement (27) requires a constant time and therefore contributes $O(n)$ to the algorithm. The conditional statement (29-37) also runs in constant time. Since it is executed for each edge of G at most twice the contribution of (28-38) in all repetitions of (26-39) is $O(\max(m, n))$.

Summing up — the total time complexity of the algorithm is $O(\max(m, n))$. Note that in a connected network $m \geq n-1$ and therefore $O(\max(m, n)) = O(m)$.

Adaptation of the algorithm for directed graphs

For directed simple graphs without loops only few changes in the implementation of the algorithm are needed depending on the interpretation of the *degree*.

In the case of in-degree (out-degree) the function `in Neighbors` must return in line 10 the next not yet visited in-neighbor (out-neighbor) and in line 28 the next not yet visited out-neighbor (in-neighbor).

If degree is defined as in-degree + out-degree, the maximum degree can be at most $2n - 2$. In this case we must provide enough space for table `bin` ($2n - 1$ elements). Function `in Neighbors` must return next not yet visited in-neighbor or out-neighbor. Note that (in and out)-neighbors are returned twice.

Example

We applied the described algorithm for cores decomposition on a network based on the Knuth's English dictionary. This network has 52652 vertices (English words having 2 to 8 characters) and 89038 edges (two vertices are adjacent, if we can get one word from another by changing, removing or inserting a letter). The obtained network is sparse: the average degree is 3.382. The program took on PC only 0.2 seconds to compute the core numbers.

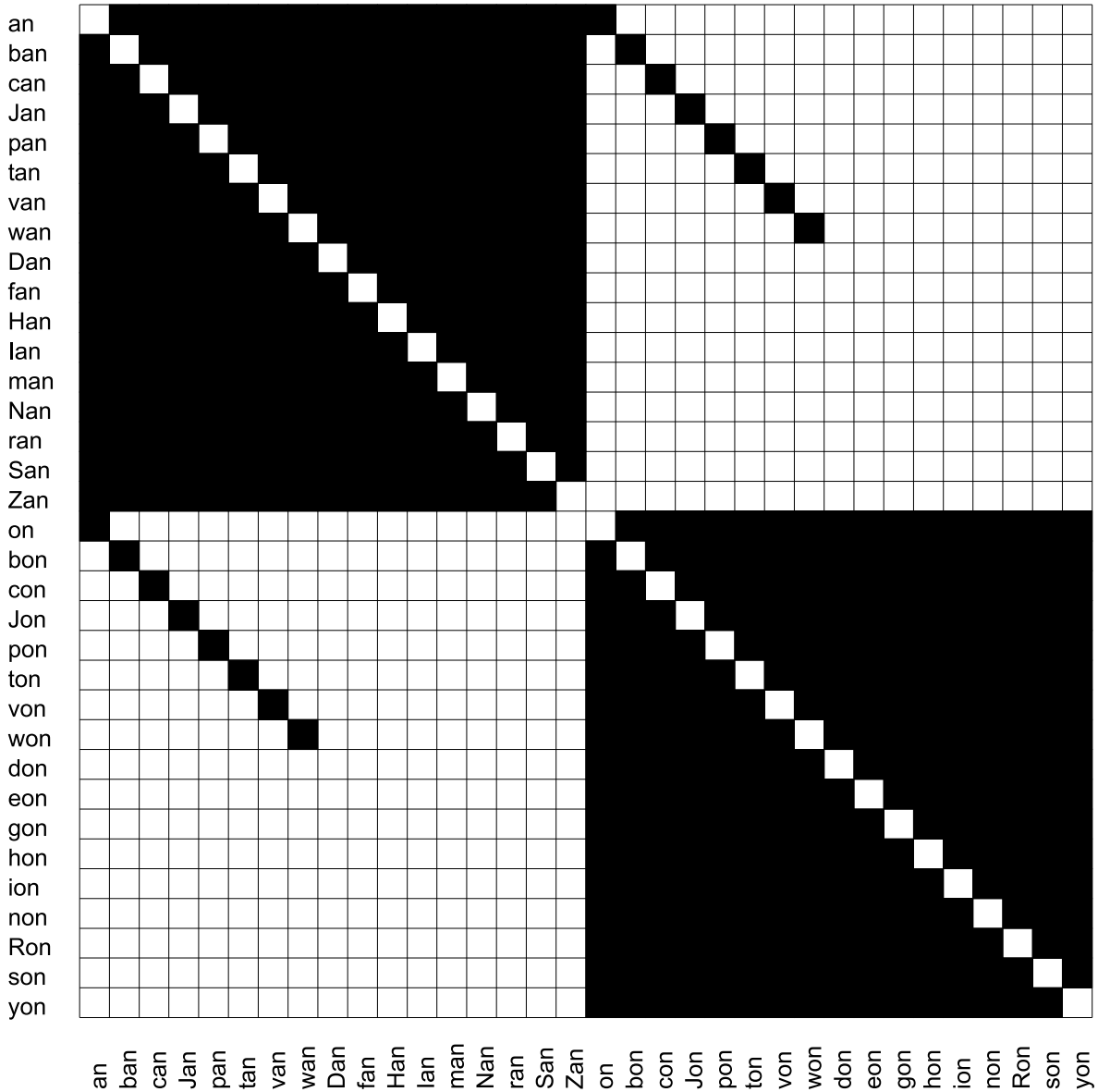
Summary results

k	vertices with core number k #	%	size of k-core #	%
25	26	0.049	26	0.049
16	34	0.065	60	0.114
15	16	0.030	76	0.144
14	59	0.112	135	0.257
13	82	0.156	217	0.412
12	200	0.380	417	0.792
11	202	0.384	619	1.176
10	465	0.883	1084	2.059
9	504	0.957	1588	3.016
8	923	1.753	2511	4.769
7	1114	2.116	3625	6.885
6	1590	3.020	5215	9.905
5	2423	4.602	7638	14.507
4	3859	7.329	11497	21.836
3	5900	11.206	17397	33.042
2	8391	15.937	25788	48.978
1	13539	25.714	39327	74.693
0	13325	25.308	52652	100.000

Vertices with core number 0 are isolated vertices. Vertices with core number 1 have only one neighbor in the network. The 25-core (main core) consists of 26 vertices, where each vertex has at least 25 neighbors inside the core (obviously this is a clique). The corresponding words are a's, b's, c's, ..., y's, z's.

The 16-core has additional 34 vertices (an, on, ban, bon, can, con, Dan, don, eon, fan, gon, Han, hon, Ian, ion, Jan, Jon, man, Nan, non, pan, pon, ran, Ron, San, son, tan, ton, van, von, wan, won, yon, Zan). There are no edges between vertices with core number 25 and vertices with core number 16. The adjacency matrix of the subgraph induced by these 34 vertices is presented on figure 1. In this matrix we can see two 17-cliques and some additional edges.

The 15-core has additional 16 vertices (ow, bow, cow, Dow, how, jow, low, mow, now, pow, row, sow, tow, vow, wow, yow). This is a clique again, because only the first letters of the words are different.


an
ban
can
Jan
pan
tan
van
wan
Dan
fan
Han
Ian
man
Nan
ran
San
Zan
on
bon
con
Jon
pon
ton
von
won
don
eon
gon
hon
ion
non
Ron
son
yon

Conclusion

The cores, because they can be efficiently determined, 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 can be built on this basis. For example, the sequence of vertices in sequential coloring can be determined by their core numbers (combined with their degrees). Cores can also be used to reveal interesting subnetworks in large networks [3, 2]:

- If it exists, a k-component is contained in a k-core.
- If it exists, a k-clique is contained in a k-core.
- $\omega(G) \leq core(G)$.

The described algorithm is implemented in program for large networks analysis **`Pajek`** (Slovene word for Spider) for Windows (32 bit) [1]. It is freely available, for noncommercial use, at its homepage:

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

References

[1] BATAGELJ, V. & MRVAR, A. (1998). Pajek – A Program for Large Network Analysis. *Connections* **21 (2)**, 47–57.

[2] BATAGELJ, V. & MRVAR, A. (2000). Some Analyses of Erdős Collaboration Graph. *Social Networks* **22**, 173–186.

[3] BATAGELJ, V., MRVAR, A. & ZAVERŠNIK, M. (1999). Partitioning approach to visualization of large graphs. In KRATOCHVÍL, Jan (ed.). Proceedings of 7th International Symposium on Graph Drawing, September 15-19, 1999, Štiřín Castle, Czech Republic. (Lecture notes in computer science, 1731). Berlin [etc.]: Springer, 90–97.

[4] GAREY, M. R. & JOHNSON, D. S. (1979). *Computer and intractability*. San Francisco: Freeman.

[5] KNUTH, D. E. (1992). Dictionaries of English words. **`ftp://labrea.stanford.edu/pub/dict/`** .

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

[7] WASSERMAN, S. & FAUST, K. (1994). *Social Network Analysis: Methods and Applications*. Cambridge: Cambridge University Press.