Adv Data Anal Classif (2011) 5:129–145
DOI 10.1007/s11634-010-0079-y

REGULAR ARTICLE

Fast algorithms for determining (generalized) core groups in social networks

Vladimir Batagelj · Matjaž Zaveršnik

Received: 3 March 2010 / Revised: 22 August 2010 / Accepted: 4 October 2010 /
Published online: 5 November 2010

Abstract The structure of a large network (graph) can often be revealed by partitioning it into smaller and possibly more dense sub-networks that are easier to handle. One of such decompositions is based on "k-cores", proposed in 1983 by Seidman. Together with connectivity components, cores are one among few concepts that provide efficient decompositions of large graphs and networks. In this paper we propose an efficient algorithm for determining the cores decomposition of a given network with complexity $\mathcal{O}(m)$, where m is the number of lines (edges or arcs). In the second part of the paper the classical concept of k-core is generalized in a way that uses a vertex property function instead of degree of a vertex. For local monotone vertex property functions the corresponding generalized cores can be determined in $\mathcal{O}(m \cdot \max(\Delta, \log n))$ time, where n is the number of vertices and Δ is the maximum degree. Finally the proposed algorithms are illustrated by the analysis of a collaboration network in the field of computational geometry.

Keywords Core · Large network · Decomposition · Graph algorithm

Mathematics Subject Classification (2000) 05A18 · 05C70 · 05C85 · 05C90 · 68R10 · 68W40 · 92H30 · 92G30 · 93A15

1 Introduction

"One of the major concerns of social network analysis is identification of cohesive subgroups of actors within a network. Cohesive subgroups are subsets of actors among

V. Batagelj (✉) · M. Zaveršnik
Department of Mathematics, FMF, University of Ljubljana, Jadranska 19, 1000 Ljubljana, Slovenia
e-mail: vladimir.batagelj@fmf.uni-lj.si

M. Zaveršnik
e-mail: matjaz.zaversnik@fmf.uni-lj.si

whom there are relatively strong, direct, intense, frequent, or positive ties" (Wasserman and Faust 1994, p. 249). Several concepts were introduced to formally describe cohesive groups of vertices or subgraphs: cliques, n-cliques, n-clans, n-clubs, k-plexes, k-cores, lambda sets, ... For most of them it turns out that they are algorithmically difficult [NP hard (Garey and Johnson 1979) or of at least quadratic time complexity] to determine, but for k-cores a very efficient algorithm exists that is described in detail in Sect. 3 and in Appendix of this paper.

In the second part of the paper we discuss the question: can the notion of a core be generalized so that cores can be efficiently determined using essentially the same approach as for ordinary cores? We present a generalization of the notion of a core of a graph based on a vertex property function. It turns out that for local monotone vertex property functions the corresponding cores can be determined in $\mathcal{O}(m \cdot \max(\Delta, \log n))$ time, where m is the number of edges, n is the number of vertices and Δ is the maximum degree.

For illustration the proposed algorithms were used for analysis of a collaboration network in the field of computational geometry with $n = 7{,}343$ vertices and $m = 11{,}898$ edges.

This paper is based on our preprints `arXiv:0310049v1` and `arXiv: 0202039v1`.

2 Cores

The notion of a core was introduced by Seidman (1983).

Let $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ be a graph. $\mathcal{V}$ is the set of *vertices* and $\mathcal{L}$ is the set of *lines* (*edges* or *arcs*). We will denote $n = |\mathcal{V}|$ and $m = |\mathcal{L}|$. The degree $\deg(v)$ of vertex v equals to the number of lines having vertex v as their endpoint. Let k be an integer. A subgraph $\mathcal{H}_k = (\mathcal{C}, \mathcal{L}|\mathcal{C})$ induced by the subset of vertices $\mathcal{C} \subseteq \mathcal{V}$ is called a *k-core* or a *core of order k* iff $\deg_{H_k}(v) \geq k$, for all $v \in \mathcal{C}$, and $\mathcal{H}_k$ is a maximum subgraph with this property—it cannot be augmented without loosing this property. The *core number* $\mathrm{core}(v)$ of vertex v is the highest order of a core that contains this vertex. The core of maximum order is also called the *main* core. Its core number is denoted by $\mathrm{core}(\mathcal{G})$.

In this definition, instead of the degree $\deg(v)$, we could also use: in-degree, out-degree, in-degree + out-degree, ... determining different types of cores.

Figure 1 illustrates the decomposition of a given graph with 21 vertices into cores of order 0, 1, 2 and 3. Note that the core of order 2 has two connected components. From this figure we can see the following basic properties of cores:

- The cores are nested: $i < j \implies \mathcal{H}_j \subseteq \mathcal{H}_i$, where $\mathcal{H}_i$ denotes the core of order i;
- Cores are not necessarily connected subgraphs.
- A *cluster* is here any nonempty subset of the set of vertices $\mathcal{V}$. A set $\mathbb{H} = \{\mathcal{C}_i : i \in I\}$ of clusters is a *hierarchy* iff $\mathcal{C}_i \cap \mathcal{C}_j \in \{\emptyset, \mathcal{C}_i, \mathcal{C}_j\}$, for all $i, j \in I$.
  - The set $\{\mathcal{C}_k : \mathcal{H}_k = (\mathcal{C}_k, \mathcal{L}_k)$ is a k-core, $k = 1, 2, \ldots, k_{\max}\}$ is a hierarchy, the 'cores hierarchy'.
  - Let $\mathrm{Con}(\mathcal{H})$ denote the set of (weakly) connected components of the graph $\mathcal{H}$. The set $\{\mathcal{C} : (\mathcal{C}, \mathcal{L}|\mathcal{C}) \in \mathrm{Con}(\mathcal{H}_k), k = 1, 2, \ldots, k_{\max}\}$ is a hierarchy.

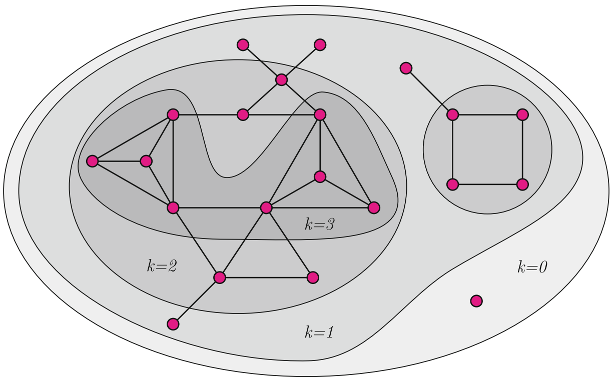

Fig. 1 0-, 1-, 2- and 3-core

In other words: the family of all cores and the family of all their connected components both generate a hierarchy of classes (clusters) of vertices.

3 Cores algorithm

Our algorithm for determining the cores hierarchy is based on the following property (Batagelj et al. 1999):

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

Note that when deleting a vertex the degrees of its neighbors decrease.

The outline of the algorithm is as follows:

Algorithm 1 In the algorithm the core number of vertex v, core(v), is represented by the table element $core[v]$, and its degree by the table element $degree[v]$.

```
INPUT: graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ represented by lists of neighbors $Neighbors(v)$ for
        each vertex $v$
OUTPUT: table $core$ with core number $core[v]$ for each vertex $v$

1.1      compute the degrees of vertices;
1.2      order the set of vertices $\mathcal{V}$ in increasing order of their degrees;
2        for each $v \in \mathcal{V}$ 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 $\mathcal{V}$ accordingly
                 end
         end;
```

The block of statements 2.2–2.2.1.2 describes the effect of deletion of the vertex v and all lines incident with it. Note that the order used in the line 2 is changed at each step by the line 2.2.1.2.

In the refinements of the algorithm we have to provide efficient implementations of steps 1.2 and 2.2.1.2. One such implementation is described in Appendix. We shall show that the described algorithm runs in time $\mathcal{O}(\max(m, n))$.

4 Example: cores of a "computational geometry" network

To illustrate the use of cores we applied the described Algorithm 1 for cores decomposition to the *authors collaboration network* based on the BibTEX bibliography (Beebe 2002) obtained from the *Computational Geometry Database* `geombib`, version February 2002 (Jones 2002). Using a simple program written in programming language Python, the BibTEX data were transformed into the corresponding network, and output to the file in Pajek format. There are 9,072 authors (vertices) dealing with computational geometry and publishing sometimes joint papers. There is an edge between two authors iff they wrote a joint paper, which results in 22,577 edges. If there are several joint papers we may replace multiple edges by a single one with their multiplicity as its weight, which results in 13,567 edges in a simple network.

The problem with the obtained network was that, because of nonstandardized writing of the author's name, it contains several vertices corresponding to the same author. For example: R.S. Drysdale, Robert L. Drysdale, Robert L. Scot Drysdale, R.L. Drysdale, S. Drysdale, R. Drysdale, and R.L.S. Drysdale; or: Pankaj K. Agarwal, P. Agarwal, Pankaj Agarwal, and P.K. Agarwal—here it is easy to guess that they belong to the same person; but an 'insider' information is needed to know that Otfried Schwarzkopf and Otfried Cheong are the same person. Also, no provision is made in the database to distinguish two persons with the same name. We manually produced the name equivalence partition and then shrank (in Pajek) the network according to it. The reduced simple network contains 7,343 vertices and 11,898 edges. It is a sparse network—its average degree is $2m/n = 3.24$. The reduced network is available for a download on Pajek's homepage among other network data (Batagelj 2004).

On the reduced network we applied our core algorithm. In Table 1 the summary results are presented.

Vertices with core number 0 are isolated vertices—noncollaborating authors. The 21-core (main core) consists of 22 vertices, where each vertex has 21 neighbors inside the core—obviously this core is a clique.

In Fig. 2 the subgraph induced by vertices in 10-core and higher is presented. It has 133 vertices. They have different colors (grey levels) indicating their core numbers. The picture was obtained using the Kamada–Kawai graph drawing procedure. Afterward some vertices were slightly repositioned to improve the readability of labels. The main core consists of the darkest vertices in the middle lower part of the picture: M.W. Bern, D. Eppstein, T.K. Dey, P. Plassmann, N. Amenta, H. Edelsbrunner, J. Harer, J. Hass, D. Letscher, E. Sedgwick, G. Lerman, D. Zorin, A. Hicks, J. Weeks, C. Grimm, C.K. Johnson, J.S. Snoeyink, P.K. Agarwal, L.J. Guibas, C.-K. Yap, D.P. Dobkin, and L.P. Chew. It is a clique. To it, lower level cores are attached. Detailed

Table 1 Core numbers distribution: the Core is the core number of vertices, Freq is the number of vertices with that core number, and Representative is an author from the corresponding set

Core	Freq	CumFreq%	Representative
0	1,185	16.1378	N. Bourbaki
1	2,218	46.3435	S. Kambhampati
2	1,714	69.6854	G. Bilardi
3	1,023	83.6171	Y.I. Yoon
4	503	90.4671	B.B. Kimia
5	248	93.8445	C.A. Duncan
6	122	95.5059	T.M. Murali
7	126	97.2218	J.M. Kleinberg
8	34	97.6849	F.F. Yao
9	37	98.1888	K.R. Lee
10	20	98.4611	H. Alt
11	52	99.1693	M. Flickner
13	1	99.1829	M.H. Overmars
14	7	99.2782	M. Sharir
15	14	99.4689	N.M. Amato
16	17	99.7004	D. White
21	22	100.0000	L.J. Guibas
Sum	7,343		

inspection of the cut at level 15 reveals that also vertices with core number 15 form a clique of order 14. This clique is linked with many ties to the subgroup {P.K. Agarwal, D.P. Dobkin, L.J. Guibas, C.-K. Yap, H. Edelsbrunner, D. Eppstein, J.S. Snoeyink, L.P. Chew and M.W. Bern} of the main core. The remaining members of the main core are collaborating mainly inside the main core. Two clusters with core numbers 16 and 11 (again, both are cliques) are linked to the main group by the 'liaison' authors (S.A. Mitchell and D.T. Lee).

Note also, that the positions determined by cores are partially different from the positions suggested by the top of the degree distribution (author and number of different co-authors): L.J. Guibas 102, P.K. Agarwal 98, J.S. Snoeyink 91, H. Edelsbrunner 90, M.H. Overmars 88, M. Sharir 87, J. O'Rourke 85, R. Tamassia 79, J.S.B. Mitchell 76, C.-K. Yap 76, E. Welzl 74, D.P. Dobkin 73, G.T. Toussaint72, M.T. Goodrich 70, K. Mehlhorn 69, R.E. Tarjan 69. Not all of them are in the main core.

The cores do not take into account how many papers two authors have in common. Therefore we elaborate, in the next section, a generalized core concept that allows also to consider values of lines and vertices.

5 Generalized cores

The concept of 'core' based on the classical degree concept can be generalized to consider other *properties* of (groups of) vertices. In network analysis an approach to the identification of important elements or parts of networks is to express the importance of a vertex as a vertex property function p.

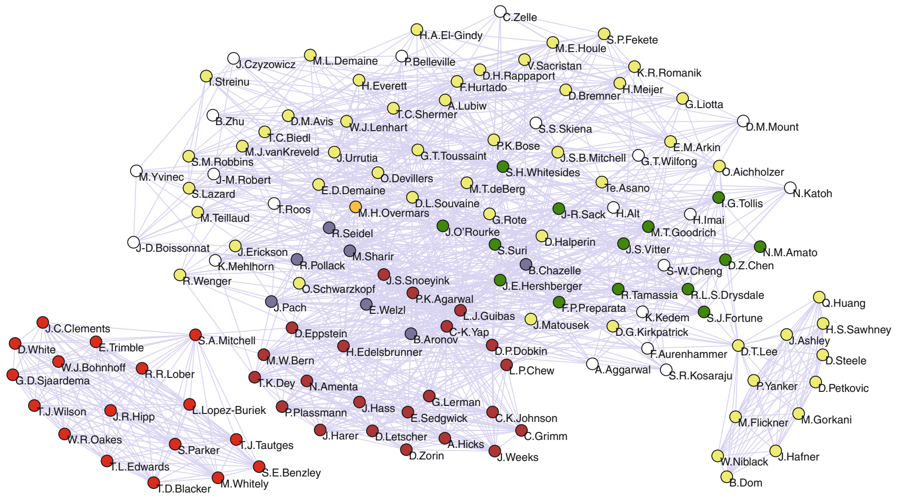

Fig. 2 The 10-core of the "Computational geometry" network

5.1 Examples of vertex property functions

Let $\mathcal{N} = (\mathcal{V}, \mathcal{L}, w)$ be a *network* where $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ is a graph and $w : \mathcal{L} \to \mathbb{R}$ is a function assigning values to lines. A *vertex property function* on $\mathcal{N}$, or a *p-function* for short, is a function $p(v, \mathcal{C})$, $v \in \mathcal{V}$, $\mathcal{C} \subseteq \mathcal{V}$ with real values. The subset $\mathcal{C}$ determines a part of the network on which the value $p(v, \mathcal{C})$ is computed.

Let $N(v)$ denote the set of neighbors of the vertex v in the graph $\mathcal{G}$, $N^+(v) = N(v) \cup \{v\}$, and $N(v, \mathcal{C}) = N(v) \cap \mathcal{C}$ for a vertex set $\mathcal{C} \subseteq \mathcal{V}$.

Here are some examples of p-functions. Functions p_1, p_2, p_3 and p_4 are used in classical types of cores.

- $p_1(v, \mathcal{C}) = \deg(v, \mathcal{C})$
- $p_2(v, \mathcal{C}) = \mathrm{indeg}(v, \mathcal{C})$
- $p_3(v, \mathcal{C}) = \mathrm{outdeg}(v, \mathcal{C})$
- $p_4(v, \mathcal{C}) = \mathrm{indeg}(v, \mathcal{C}) + \mathrm{outdeg}(v, \mathcal{C})$
- $p_5(v, \mathcal{C}) = \frac{\deg(v,\mathcal{C})}{\deg(v)}$, if $\deg(v) > 0$; $= 0$, otherwise
- relative density
  $p_6(v, \mathcal{C}) = \dfrac{\deg(v,\mathcal{C})}{\max_{u \in N(v)} \deg(u)}$, if $\deg(v) > 0$; $= 0$, otherwise
- diversity of neighbors
  $p_7(v, \mathcal{C}) = \max_{u \in N(v,\mathcal{C})} \deg(u) - \min_{u \in N(v,\mathcal{C})} \deg(u)$
- diversity of neighborhoods
  $p_8(v, \mathcal{C}) = \max_{u \in N^+(v,\mathcal{C})} \deg(u) - \min_{u \in N^+(v,\mathcal{C})} \deg(u)$
- clustering coefficient: for an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $K(\mathcal{U})$ denotes the complete graph on a vertex set $\mathcal{U} \subseteq \mathcal{U}$ and $\mathcal{E}(N)$ is the set of edges from $\mathcal{E}$ between vertices from N
  $p_9(v, \mathcal{C}) = \frac{|\mathcal{E}(N(v,\mathcal{C}))|}{|\mathcal{E}(K(N(v)))|}$, if $\deg(v) > 1$; $= 0$ otherwise

- corrected clustering coefficient: $\Delta(\mathcal{G}) = \max_{u \in \mathcal{V}} \deg(u)$
  $p_{10}(v, \mathcal{C}) = \frac{\deg(v)}{\Delta(\mathcal{G})} p_9(v, \mathcal{C})$
- $p_{11}(v, \mathcal{C}) = \sum_{u \in N(v,\mathcal{C})} w(v, u)$, where $w : \mathcal{L} \to \mathbb{R}_0^+$
- $p_{12}(v, \mathcal{C}) = \max_{u \in N(v,\mathcal{C})} w(v, u)$, where $w : \mathcal{L} \to \mathbb{R}$
- $p_{13}(v, \mathcal{C}) =$ number of cycles of length k through vertex v passing through vertices from $\mathcal{C}$ only
- average weight
  $p_{14}(v, \mathcal{C}) = \frac{1}{|N(v,\mathcal{C})|} \sum_{u \in N(v,\mathcal{C})} w(v, u)$, if $N(v, \mathcal{C}) \neq \emptyset$; $= 0$, otherwise

5.2 Generalized cores and their properties

The simplest way to identify interesting parts of a network is to use the *vertex cut* for a selected threshold t—the corresponding parts are connected components of the induced subnetwork on the set of vertices $\mathcal{C} \subseteq \mathcal{V}$ with property value $p(v) \geq t$

$$\mathcal{C} = \{v \in \mathcal{V} : p(v) \geq t\}$$

Note that these values need not to be at least t if p is restricted to the obtained subnetwork. This requirement leads to the notion of p-cores.

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

- $\forall v \in \mathcal{C} : t \leq p(v, \mathcal{C})$
- $\mathcal{C}$ is the maximum set with this property.

Note that an ordinary k-core is a p_1-core at level k.

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

$$\mathcal{C}_1 \subset \mathcal{C}_2 \Rightarrow \forall v \in \mathcal{V} : (p(v, \mathcal{C}_1) \leq p(v, \mathcal{C}_2))$$

In words, the value $p(v, \mathcal{C})$ of the vertex property function p for a vertex v will not decrease if we increase the underlying subset $\mathcal{C} \subseteq \mathcal{V}$.

All functions $p_1, \ldots, p_{13}$ are monotone. Function p_{14} is not monotone. For any monotone p-function we can extend the approach used for determining the ordinary cores (see Sect. 3) to the case of p-cores. The p-core $\mathcal{H}_t$ at level t can be determined by successively deleting vertices with value of p lower than t.

Algorithm 2

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

Theorem 1 *For each monotone vertex property function p Algorithm 2 determines the p-core $\mathcal{H}_t$ at level t.*

Proof The set $\mathcal{C}$ returned by the procedure evidently has the first property from the p-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, determined by sets $\mathcal{C}$ and $\mathcal{D}$. The core $\mathcal{C}$ was produced by deleting the sequence $u_1, u_2, u_3, \ldots, u_p$; and $\mathcal{D}$ by the sequence $v_1, v_2, v_3, \ldots, v_q$. Assume that $\mathcal{D} \setminus \mathcal{C} \neq \emptyset$. We will show that this leads to contradiction.

Take any $z \in \mathcal{D} \setminus \mathcal{C}$. To show that it also can be deleted we first apply the sequence $v_1, v_2, v_3, \ldots, v_q$ to get $\mathcal{D}$. Since $z \in \mathcal{D} \setminus \mathcal{C}$ it appears in the 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 $p(u_i, \mathcal{V} \setminus U_{i-1}) < t$, for all $i \in 1 \ldots p$, we have, by monotonicity of p, also $p(u_i, (\mathcal{V} \setminus \mathcal{D}) \setminus U_{i-1}) < t$, for all $i \in 1 \ldots p$. Therefore also all $u_i \in \mathcal{D} \setminus \mathcal{C}$ are deleted—$\mathcal{D} \setminus \mathcal{C} = \emptyset$—a contradiction.

Since the result of the procedure is uniquely defined and vertices outside $\mathcal{C}$ have p value lower than t, the final set $\mathcal{C}$ satisfies also the second condition from the definition of p-core—it is the p-core at level t. □

Corollary 1 *For each monotone p-function p the cores are nested*

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

Proof Follows directly from Theorem 1. Since the result is independent of the order of deletions we first determine the $\mathcal{H}_{t_1}$. In the following we eventually delete some additional vertices thus producing $\mathcal{H}_{t_2}$. Therefore $\mathcal{H}_{t_2} \subseteq \mathcal{H}_{t_1}$. □

Counter-example of a nonmonotone p-function We consider the weighted network $\mathcal{N} = (\mathcal{V}, \mathcal{L}, w)$ with vertex set $\mathcal{V} = \{a, b, c, d, e, f\}$ and the weights $w : \mathcal{L} \to \mathbb{R}_0^+$

$\mathcal{L}$	$(a:b)$	$(b:c)$	$(c:d)$	$(b:e)$	$(e:f)$
w	4	1	3	1	3

See Fig. 3. For defining p-cores we use the p-function p_{14} which is not monotone. For example, $\{c, d\} \subset \mathcal{V}$, but $p_{14}(c, \{c, d\}) = 3$ and $p_{14}(c, \mathcal{V}) = 2$.

In Fig. 3 the values of p-function are written inside the circles representing vertices. The edges are labeled by the corresponding weights. The arrows between different versions of network are labeled by deleted vertex.

The original network is a p_{14}-core at level 2. Applying Algorithm 2 to the network we have three choices for the first vertex to be deleted: b, c or e. Deleting b we get, after removing the isolated vertex a, the p-core $\mathcal{C}_1 = \{c, d, e, f\}$ at level 3. Note that the values of p in vertices c and e increased from 2 to 3.

Deleting c (or symmetrically e—we analyze only the first case) we get the set $\mathcal{C}_2 = \{a, b, e, f\}$ at level 2—the value at b increased to 2.5. In the next step we can delete either the vertex b, producing the set $\mathcal{C}_3 = \{e, f\}$ at level 3, or the vertex e, producing the p-core $\mathcal{C}_4 = \{a, b\}$ at level 4.

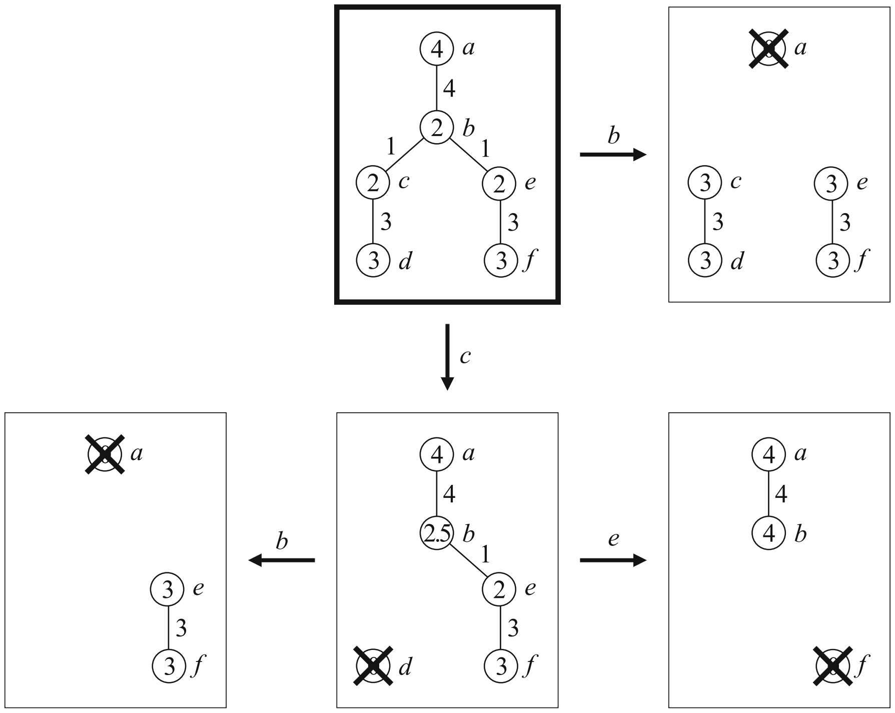

Fig. 3 Nonmonotone p-function; the result depends on the order of deletions

As we see, the result of the algorithm depends on the order of deletions. We get different results depending on whether we first delete the vertex b or c (or e). The p-core at level 4 is not contained in the p-core at level 3.

6 Algorithms for computing generalized cores

6.1 Algorithm for p-core at level t

A p-function is termed *local* iff

$$p(v, \mathcal{C}) = p(v, N(v, \mathcal{C})), \quad \text{for all } v \in \mathcal{V}$$

or, in words, iff its value at v depends only on vertices in the neighborhood of v.

All p-functions from our examples are local, except p_{13} for $k \geq 4$. In the following we shall assume also that for the function p there exists a constant p_0 such that for all $v \in \mathcal{V} p(v, \emptyset) = p_0$. This holds for all functions from our examples.

Assuming that $p(v, N(v, \mathcal{C}))$ can be computed in $\mathcal{O}(\deg(v, \mathcal{C}))$ time, we show in the following that in case of a monotone and local vertex property function p there exists an Algorithm 3 that determines the p-core at level t in $\mathcal{O}(m \cdot \max(\Delta, \log n))$ time. For operations on heaps (see Cormen et al. 2001).

Algorithm 3

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

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

The step 4.2.1. can often be speeded up by updating the $p[v]$—considering the change of $p[v]$ in each step.

6.2 Determining the hierarchy of p-cores

The Algorithm 3 for determining the p-core of $\mathcal{G}$ at level t can be easily extended to produce the hierarchy (of vertex sets) of p-cores. The hierarchy is determined by the core-value assigned to each vertex—the highest level value of p-cores that contain the vertex.

Algorithm 4

INPUT: graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$ represented by lists of neighbors,
monotone vertex property function p
OUTPUT: table $core$ with core-value for each vertex

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

Theorem 2 *Assume that $p(v, N(v, \mathcal{C}))$ can be computed in $\mathcal{O}(\deg(v, \mathcal{C}))$ time. Then for a monotone and local vertex property function p Algorithm* 4 *determines the p-core hierarchy in $\mathcal{O}(m \cdot \max(\Delta, \log n))$ time.*

Proof Let us assume that P is a maximum time needed for computing the value of $p(v, \mathcal{C})$ for all $v \in \mathcal{V}, \mathcal{C} \subseteq \mathcal{V}$. Then the complexity of statements 1.–3. in Algorithm 4 is $T_{1-3} = \mathcal{O}(n) + \mathcal{O}(Pn) + \mathcal{O}(n \log n) = \mathcal{O}(n \cdot \max(P, \log n))$. Let us now look at the body of the while loop. Since at each repetition of the body the size of the set $\mathcal{C}$ is decreased by 1 there are at most n repetitions. Statements 4.1 and 4.2 can be implemented to run in constant time, thus contributing $T_{4.1,4.2} = \mathcal{O}(n)$ to the loop. In all combined repetitions of the while and for loops each line is considered at most once. Therefore the body of the for loop (statements 4.3.1 and 4.3.2) is executed at most m times—contributing at most $T_{4.3} = m \cdot (P + \mathcal{O}(\log n))$ to the while loop. Often the value $p(v, N(v, \mathcal{C}))$ can be updated in constant time—$P = \mathcal{O}(1)$. Summing up all the contributions we get the total time complexity of the algorithm $T = T_{1-3} + T_{4.1,4.2} + T_{4.3} = \mathcal{O}(m \cdot \max(P, \log n))$. For a local p-function, for which the value of $p(v, N(v, \mathcal{C}))$ can be computed in $\mathcal{O}(\deg(v, \mathcal{C}))$, we have $P = \mathcal{O}(\Delta)$. □

Using a Fibonacci heap (Cormen et al. 2001) the *update_heap* operation can be done in constant time, improving the time bound to $\mathcal{O}(m\Delta + n \log n)$.

The described algorithms for generalized cores are partially implemented in the program for large networks analysis **Pajek** (Slovene word for Spider) for Windows (Batagelj and Mrvar 2003). It is freely available, for noncommercial use, at the homepage: http://pajek.imfm.si/. A stand-alone implementation of the algorithms in C is available at http://www.educa.fmf.uni-lj.si/datana/pub/networks/cores/.

7 Example

As an example we consider again the "Computational Geometry" network. We replaced multiple edges between a pair of authors with a single edge with their multiplicity as weight. Using the ordinary, classical vertex property function $p_1 = \deg$, the corresponding p_1-cores give us the answer to the question: who are the authors that have co-authored works with the largest number of other such authors. To answer the question: who are the authors that have co-authored the most works with other important authors, we have to use an appropriate vertex property function. Such a function is the function $p_S \equiv p_{11}$ that counts the total number of works an author from the core wrote with other authors from the core. On the obtained network we applied Algorithm 4 for determining p_S-cores. Figure 4 shows the p_S-core at level 46. This level was determined by considering the frequency distribution of values of the function p_S. The line weights are represented by their thickness.

The p_S-core gives us another view on the collaboration in the field of computational geometry. Only some of the authors from the main deg-core belong also to p_S-core. Besides the main group at the middle left side some pairs of authors appear: M. Bern, D. Eppstein; E. Arkin, J. Mitchell; J. Urrutia, J. Czyzowicz; P. Bose, G. Toussaint; C. Icking, R. Klein; and three groups: Tamassia's group (R. Tamassia, G. Liotta, I. Tollis, A. Garg, G. diBattista, L. Vismara, M. Goodrich, J. Vitter), Canadian group (M. Teillaud, M. Yvinec, O. Devillers, J. Boissonnat) and Smid's group (M. Smid, P. Gupta, J. Majhi, R. Janardan, J. Schwerdt). F. Preparata is a kind of broker between Tamassia's and Canadian groups.

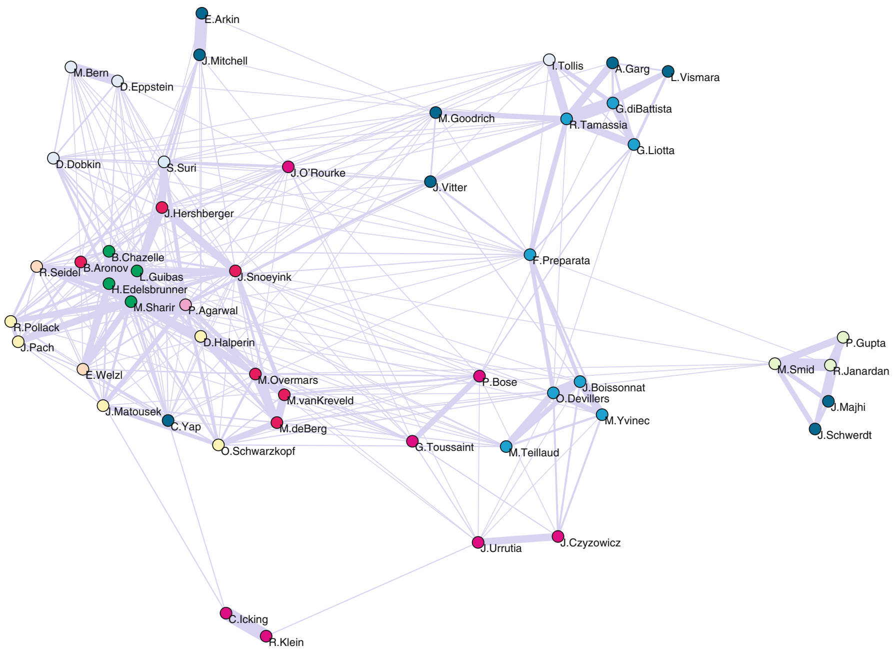

Fig. 4 p_S-core of "Computational geometry" network at level 46

8 Practical applications and extensions

The cores, because they can be efficiently determined, are one among some few concepts that provide us with meaningful decompositions of large networks. We expect that also other approaches for analysing large networks can be built on this basis. For example, the sequence of vertices in a sequential coloring (Welsh and Powell 1967) can be determined by the descending order of their core numbers (combined with their degrees). We obtain on this basis the following bound on the chromatic number of a given graph $\mathcal{G}$

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

Cores can also be used to localize the search for interesting subnetworks in large networks (Batagelj et al. 1999; Batagelj 2000) considering the properties such as:

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

The notion of cores can be extended as (k_1, k_2)-cores also to two-mode (bipartite) networks (Ahmed et al. 2007).

The proposed algorithms have various other applications, e.g., for the analysis of protein interaction networks and biochemical networks (Wuchty and Almaas 2005; Schwartz and Nacher 2009); visualization of large networks: representation by iconic fingerprints (LaNet-vi 2009; Alvarez-Hamelin et al. 2008; Beiro et al. 2008) and planar

```
procedure cores(var g: graph; var deg: tableVert);
var
   n, d, md, i, start, num: integer;
   v, u, w, du, pu, pw: integer;
   vert, pos: tableVert;
   bin: tableDeg;
begin
   n := size(g);  md := 0;
   for v := 1 to n do begin
      d := 0;  for u in Neighbors(g, v) do inc(d);
      deg[v] := d;  if d > md then md := d;
   end;
   for d := 0 to md do bin[d] := 0;
   for v := 1 to n do inc(bin[deg[v]]);
   start := 1;
   for d := 0 to md do begin
      num := bin[d];
      bin[d] := start;
      inc(start, num);
   end;
   for v := 1 to n do begin
      pos[v] := bin[deg[v]];
      vert[pos[v]] := v;
      inc(bin[deg[v]]);
   end;
   for d := md downto 1 do bin[d] := bin[d-1];
   bin[0] := 1;
   for i := 1 to n do begin
      v := vert[i];
      for u in Neighbors(g, v) do begin
         if deg[u] > deg[v] then begin
            du := deg[u];   pu := pos[u];
            pw := bin[du];  w := vert[pw];
            if u <> w then begin
               pos[u] := pw;  vert[pu] := w;
               pos[w] := pu;  vert[pw] := u;
            end;
            inc(bin[du]);  dec(deg[u]);
         end;
      end;
   end;
end;
```

Fig. 5 The cores algorithm for simple undirected graphs

core decomposition (Batagelj et al. 2010); recommendation systems and folksonomies (Jäschke 2007; Eisterlehner et al. 2009) and auctions (Wang and Chiu 2008). These applications also give importance to the theoretical discussions of the properties and role of cores (Dorogovtsev et al. 2006; Janson 2008). The fast core Algorithm 1 is included in several network analysis libraries and tools: **Pajek**, R (sna, igraph), Complex Systems Toolbox/Scilab, MatlabBGL, Netminer, SNAP, NetworkWorkbench and NetworkX.

Acknowledgments This work was supported by the Ministry of Education, Science and Sport of Slovenia, Project J1-8532. We wish to thank the anonymous reviewers and the editor Hans-Hermann Bock for their helpful suggestions on improving the paper during the review process.

Appendix A: Details on the implementation of the cores Algorithm 1

In Fig. 5 we describe an implementation of Algorithm 1 in a Pascal like language for the case of simple undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{E}$ is the set of edges. We use some nonstandard functions:

function	meaning
`dec(a)`	`a := a - 1`
`inc(a)`	`a := a + 1`
`inc(a, b)`	`a := a + b`

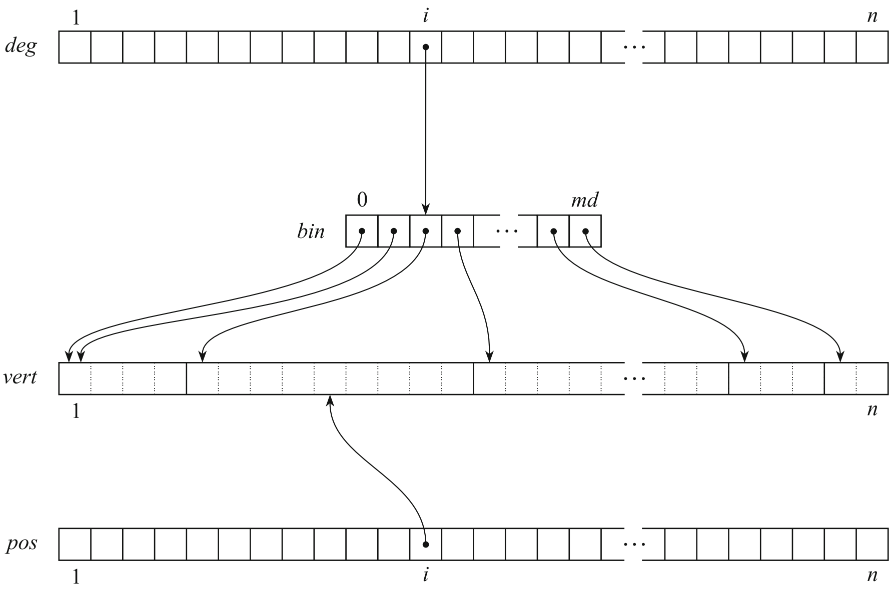

Fig. 6 Arrays

The structure `graph` is used to represent a given graph $\mathcal{G}$. We will not describe the structure into detail, because there are several possibilities how to implement it. We assume that the vertices of $\mathcal{G}$ are numbered from 1 to n. The user has also to provide function `size`, which returns the number of vertices in the given graph, and function `in Neighbors`, which returns the next not yet visited neighbor of a given vertex in the given graph. Using an adequate representation of graph $\mathcal{G}$ (lists of neighbors) we can implement both functions to run in constant time.

Two types of integer arrays (`tableVert` and `tableDeg`) are also introduced. Both of them are of length at least n. The only difference is how we index their elements. We start with index 1 in `tableVert` and with index 0 in `tableDeg`.

The Algorithm 1 is implemented by the procedure `cores`. Its input is a graph $\mathcal{G}$, represented by the variable `g` of type `graph`, and the output is array `deg` of type `tableVert` containing the core number for each vertex of graph $\mathcal{G}$.

We also need (lines 03–06) some integer variables and three additional arrays (see Fig. 6). The array `vert` contains the set of vertices, sorted by their degrees. The positions of vertices in array `vert` are stored in array `pos`. The array `bin` contains for each possible degree the position of the first vertex of that degree in array `vert`.

In a real implementation of the proposed algorithm dynamically allocated arrays should be used. To simplify our description of the algorithm we replaced them by static.

At the beginning we have to initialize some local variables and arrays (lines 08–12). First we determine `n`, the number of vertices of graph `g`. Then *we compute its degree* *for each vertex* v *in graph* g and store it into array deg. Simultaneously we also compute the maximum degree md.

Since the values of degrees are integers from the interval $0 \ldots n-1$, we can *sort the vertices in increasing order of their degrees* in linear time using a variant of bin-sort (lines 13–25).

First we count (lines 13–14) how many vertices will be in each bin (bin consists of vertices with the same degree). The bins are numbered from 0 to md. From the bin sizes we can determine (lines 15–20) starting positions of bins in the array vert. Bin 0 starts at position 1, while other bins start at position, equal to the sum of starting position and size of the previous bin. To avoid an additional array we used the same array (bin) to store the starting positions of bins. Now we can put (lines 21–25) vertices of the graph $\mathcal{G}$ into the array vert. For each vertex we know to which bin it belongs and what is the starting position of that bin. So we can put the current vertex to the proper place, remember its position in the table pos, and increase the starting position of the bin we used. The vertices are now sorted by their degrees.

In the final step of the initialization phase we have to *recover the starting positions of the bins* (lines 26–27). We increased them several times in the previous step, when we put vertices into corresponding bins. It is obvious, that the changed starting position is the original starting position of the next bin. To restore the right starting positions we have to shift the values in array bin for one position to the right. We also have to reset the starting position of the bin 0 to value 1.

The *cores decomposition*, implementing the for each loop from the algorithm described in Sect. 3, is done in the main loop (lines 28–41) that runs over all vertices v of graph g in the order, determined by the table vert. The core number of the current vertex v is the current degree of that vertex. This number is already stored in table deg. For each neighbor u of vertex v with higher degree we have to decrease its degree by 1 and move it for one bin to the left. Moving vertex u for one bin to the left is operation, which can be done in constant time. First we have to swap the vertex u and the first vertex in the same bin. We also have to swap their positions in the array pos. Finally we increase the starting position of the bin (we increase the previous and reduce the current bin for one element).

A.1. Time complexity

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

To compute (lines 08–12) the degrees of all vertices we need time $\mathcal{O}(\max(m, n))$ since we have to consider each line at most twice. The *bin sort* (lines 13–27) consists of five loops of size at most n with constant time $\mathcal{O}(1)$ bodies—therefore it runs in time $\mathcal{O}(n)$.

The statement (line 29) requires a constant time and therefore contributes $\mathcal{O}(n)$ to the algorithm. The conditional statement (lines 31–39) also runs in constant time. Since it is executed for each edge of $\mathcal{G}$ at most twice the contribution of (lines 30–40) in all repetitions of (lines 28–41) is $\mathcal{O}(\max(m, n))$.

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

A.2. Adaption 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` in line 10 must return the next not yet visited in-neighbor (out-neighbor), and the function `in Neighbors` in line 30 must return the next not yet visited out-neighbor (in-neighbor).

If the degree is defined as in-degree + out-degree, the maximum degree can be at most $2n-2$. In this case we have to increase the size of table `bin` to $2n-1$ elements. The function `in Neighbors` must return the next not yet visited in-neighbor or out-neighbor.

References

Ahmed A, Batagelj V, Fu X, Hong S-H, Merrick D, Mrvar A (2007) Visualisation and analysis of the internet movie database. In: Proceedings of the Asia-Pacific symposium on visualisation (APVIS2007), Sydney, NSW, Australia, 5–7 February 2007. IEEE, New York, 17–24

Alvarez-Hamelin JI, Dall'asta L, Barrat A, Vespignani A (2008) K-core decomposition of internet graphs: hierarchies, selfsimilarity and measurement biases. Netw Heterog Media 3(2):371–393

Batagelj V, Mrvar A (2003) Pajek—analysis and visualization of large networks. In: Jünger M, Mutzel P (eds) Graph drawing software. Springer, Berlin, pp 77–103. http://pajek.imfm.si

Batagelj V (2004) Pajek datasets: Geom. http://vlado.fmf.uni-lj.si/pub/networks/Data/Collab/Geom.htm

Batagelj V, Mrvar A (2000) Some analyses of Erdős collaboration graph. Soc Netw 22:173–186

Batagelj V, Mrvar A, Zaveršnik M (1999) Partitioning approach to visualization of large graphs. In: KratochvÍl J (ed) Proceedings of 7th international symposium on graph drawing, 15–19 September 1999, Štiřín Castle, Czech Republic (Lecture notes in computer science, vol. 1731). Springer, Berlin, pp 90–97

Batagelj V, Brandenburg FJ, Didimo W, Liotta G, Palladino P, Patrignani M (2010) Visual analysis of large graphs using (X;Y)-clustering and hybrid visualizations. In: IEEE Pacific visualization 2010 (PacVis '10). IEEE, New YorK, pp 209–216

Beebe NHF (2002) Nelson H. F. Beebe's bibliographies page. http://www.math.utah.edu/~beebe/bibliographies.html

Beiro MG, Alvarez-Hamelin JI, Busch JR (2008) A low complexity visualization tool that helps to perform complex systems analysis. New J Phys 10:125003, 1–18

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge

Dorogovtsev SN, Goltsev AV, Mendes JFF (2006) k-Core architecture and k-core percolation on complex networks. Phys D Nonlinear Phenom 224(1–2):7–19

Eisterlehner F, Hotho A, Jäschke R (eds) (2009) Proceedings of ECML PKDD discovery challenge 2009 (DC09). http://www.kde.cs.uni-kassel.de/ws/dc09/papers/proceedings.pdf

Garey MR, Johnson DS (1979) Computer and intractability. Freeman, San Francisco

Janson S, Luczak MJ (2008) Asymptotic normality of the k-core in random graphs. Ann Appl Probab 18(3):1085–1137

Jäschke R, Marinho L, Hotho A, Schmidt-Thieme L, Stumme G (2007) Tag recommendations in folksonomies. Lecture notes in computer science, vol 4702. Springer, Berlin, pp 506–514

Jones B (2002) Computational geometry database. http://compgeom.cs.uiuc.edu/~jeffe/compgeom/biblios.html, ftp://ftp.cs.usask.ca/pub/geometry/

LaNet-vi (2009) Large network visualization tool. http://xavier.informatics.indiana.edu/lanet-vi/

Seidman SB (1983) Network structure and minimum degree. Soc Netw 5:269–287

Schwartz J-M, Nacher JC (2009) Local and global modes of drug action in biochemical networks. BMC Chem Biol 9:4–114

Wang J-C, Chiu C-C (2008) Recommending trusted online auction sellers using social network analysis. Expert Syst Appl Int J Arch 34(3):1666–1679

Wasserman S, Faust K (1994) Social network analysis: methods and applications. Cambridge University Press, Cambridge

Welsh DJA, Powell MB (1967) An upper bound for the chromatic number of a graph and its application to timetabling problems. Comput J 10(1):85–86

Wuchty S, Almaas E (2005) Peeling the yeast protein network. Proteomics 5:444–449