An $O(m)$ Algorithm for Cores Decomposition of Networks

Vladimir Batagelj, Matjaž Zaveršnik
Department of Mathematics, University of Ljubljana, Slovenia
vladimir.batagelj@uni-lj.si
matjaz.zaversnik@fmf.uni-lj.si

April 24, 2002 / September 1, 2002

Abstract

The structure of large networks can be revealed by partitioning them to smaller parts, which are easier to handle. One of such decompositions is based on k–cores, proposed in 1983 by Seidman. In the paper an efficient, $O(m)$, m is the number of lines, algorithm for determining the cores decomposition of a given simple network is presented. An application on the authors collaboration network in computational geometry is presented.

Keywords: core, large network, decomposition, graph algorithm

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 whom there are relatively strong, direct, intense, frequent, or positive ties” ([[1]], p. 249). Several notions were introduced to formally describe cohesive groups: 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 [[2]] or at least quadratic), but for cores a very efficient algorithm exists. We describe it in details in this paper.

2 Cores

The notion of a core was introduced by Seidman in 1983 [[3]].

Let $G = (V, L)$ be a 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 $H_k = (W, L|W)$ 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_k is the 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.

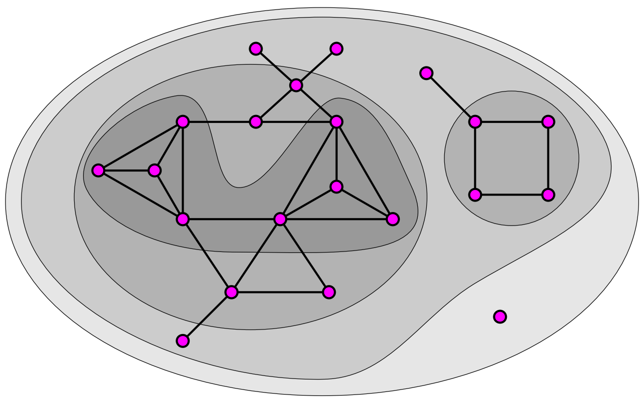

Figure 1: *0, 1, 2 and 3 core*

The degree deg(v) can be: in-degree, out-degree, in-degree + out-degree, …determining different types of cores.

In Figure 1 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.

3 Algorithm

Our algorithm for determining the cores hierarchy is based on the following property [[8]]:

> 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.

The outline of the algorithm is as follows:

```
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 begin
2.1         $core[v] := degree[v]$;
2.2         for each $u \in Neighbours(v)$ 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;
```

The block of statements 2.2.* 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.

4 Detailed Algorithm

In the Algorithm 1 we describe an implementation of the algorithm in a Pascal like language for the case of simple undirected graph $G = (V, E)$, E is the set of edges.

The structure `graph` is used to represent a given graph $G = (V, L)$. We will not describe the structure into details, because there are several possibilities, how to implement it. We assume that the vertices of 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 Neighbours`, which returns the next not yet visited neighbour of a given vertex in the given graph. Using an adequate representation of graph G (lists of neighbours) we can implement both functions to run in a constant time.

Two different types of integer arrays (`tableVert` and `tableDeg`) are also introduced. Both of them are of length 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 is implemented by the procedure `cores`. Its input is a graph G, represented by the variable `g` of type `graph`; the output is array `deg` of type `tableVert` containing the core number for each vertex of graph G.

We also need (04-07) some integer variables and three additional arrays (see Figure 2). 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 initialise some local variables and arrays (09-15). First we determine `n`, the number of vertices of graph `g`. Then ***we compute its degree for each vertex*** `v` ***in the graph*** `g` and store it into the array `deg`. Simultaneously we also compute the maximum degree `md`.

Since the values of degrees are integers from the interval 0 .. $n - 1$, we can ***sort the vertices in increasing order of their degrees*** in linear time using a variant of bin-sort (16-28).

First we count (16-17) how many vertices will be in each bin (bin consists of vertices of the same degree). The bins are numbered from 0 to `md`. From the bin sizes we can determine (18-23) starting positions of bins in the array `vert`. The bin 0 starts at position 1, while other bins start at position, equal to the sum of starting position and size of the

Algorithm 1: *The Cores Algorithm for Simple Undirected Graphs*

```
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 Neighbours(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 Neighbours(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;
               pos[w] := pu;
               vert[pu] := w;
               vert[pw] := u;
            end;
            inc(bin[du]);
            dec(deg[u]);
         end;
      end;
   end;
end;
```

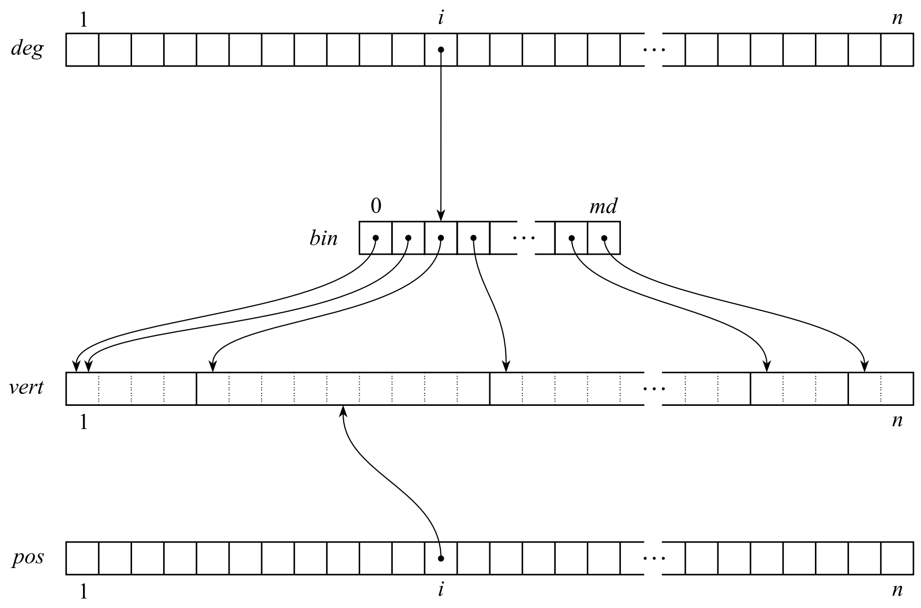

Figure 2: *Arrays*

previous bin. To avoid an additional array we used the same array (`bin`) to store the starting positions of bins. Now we can put (24-28) vertices of the graph 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 initialisation phase we have to ***recover the starting positions of the bins*** (29-31). We increased them several times in 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 section 3, is done in the main loop (32-50) that runs over all vertices `v` of the 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 neighbour `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 an operation that can be done in a 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).

4.1 Time complexity

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

To compute (09-15) 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* (16-31) consists of five loops of size at most n with constant time $O(1)$ bodies – therefore it runs in time $O(n)$.

The statement (33) requires a constant time and therefore contributes $O(n)$ to the algorithm. The conditional statement (35-48) also runs in constant time. Since it is executed for each edge of G at most twice the contribution of (34-49) in all repetitions of (32-50) 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 $O(\max(m, n)) = O(m)$.

4.2 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 Neighbours` in line 12 must return the next not yet visited in-neighbour (out-neighbour), and the function `in Neighbours` in line 34 must return the next not yet visited out-neighbour (in-neighbour).

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 Neighbours` must return the next not yet visited in-neighbour or out-neighbour.

The described algorithm is implemented in program for large networks analysis `Pajek` (Slovene word for Spider) for Windows (32 bit) [[4]]. It is freely available, for noncommercial use, at its homepage:
`vlado.fmf.uni-lj.si/pub/networks/pajek/`

5 Example

To illustrate the use of cores we applied the described algorithm for cores decomposition on the ***authors collaboration network*** based on the BibTEX bibliography [[5]] obtained from the ***Computational Geometry Database*** `geombib`, version February 2002 [[6]]. Two authors are linked with an edge, iff they wrote a common paper. 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. The obtained network has 9072 vertices (authors) and 22577 edges (common papers or books) / 13567 edges as a simple network – multiple edges between a pair of authors are replaced with a single edge.

The problem with the obtained network is that, because of non standardized 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 – that are easy to guess; 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 discern two persons with the same name.

Table 2: Core Numbers Distribution

Core	Freq	CumFreq%	Representative
0	1185	16.1378	N. Bourbaki
1	2218	46.3435	S. Kambhampati
2	1714	69.6854	G. Bilardi
3	1023	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	7343		

We manually produced the name equivalence partition and then shrank (in Pajek) the network according to it. The reduced simple network contains 7343 vertices and 11898 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.

On the reduced network we applied our core algorithm. In Table 2 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 at least 21 neighbours inside the core (obviously this is a clique).

In Figure 3 the subgraph induced by vertices in 10-core and higher is presented. It has 133 vertices. They have different colours (gray levels) indicating the order of the core. The picture was obtained using 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. To it, lower level cores are attached. Detailed 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 'liason' authors (Scott A. Mitchell and D.T. Lee).

Note also, that the positions offered by cores are partially different from the positions suggested by the top of the degree distribution (author and number of different coauthors):

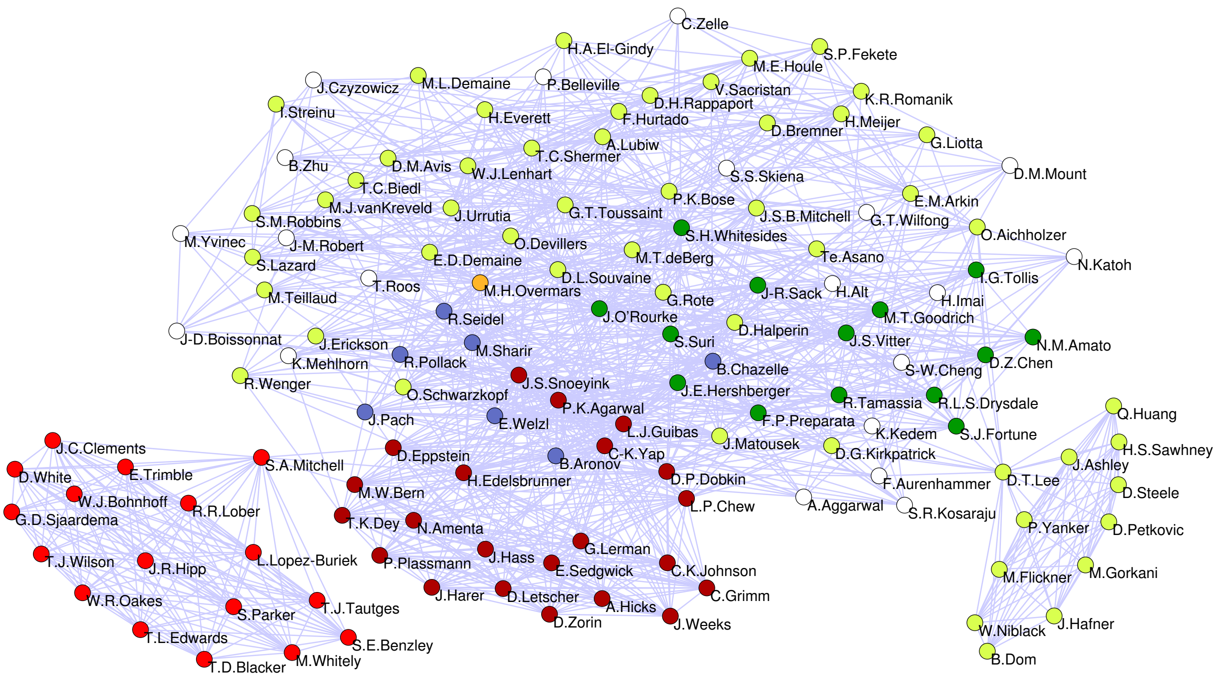

Figure 3: *Computational Geometry 10 Core*

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. Toussaint 72, M.T. Goodrich 70, K. Mehlhorn 69, R.E. Tarjan 69.

The cores do not take into account, how many papers have two authors in common. To consider also the values on lines a generalized notion of core was introduced [[7]]. Also for them an efficient (subquadratic) algorithm exists, with the time complexity $O(m \cdot \max(\Delta, \log n))$, Δ denotes the maximum degree in the network.

6 Conclusion

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

First, they can be used for quick identification of important parts of a network [[8, 9]]. Second, since some other types of subgraphs (for example: k-cliques, k-connected components) are contained in the k-core, it can be used to speed-up the corresponding search algorithms. And third, the ordering determined by core numbers can prove useful in some heuristic algorithms – for example, the sequence of vertices in sequential colourings can be determined by their core numbers (combined with their degrees).

Acknowledgments

This work was supported by the Ministry of Education, Science and Sport of Slovenia, Project J1-8532. It is a detailed version of the part of the talk presented at *Recent Trends in Graph Theory, Algebraic Combinatorics, and Graph Algorithms*, September 2001, Bled, Slovenia.

References

[1] Wasserman, S. and Faust, K. (1994) *Social Network Analysis: Methods and Applications*, Cambridge University Press, Cambridge.

[2] Garey, M. R. and Johnson, D. S. (1979) *Computer and intractability*, Freeman, San Francisco.

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

[4] Batagelj, V. and Mrvar, A. (1998) Pajek – A Program for Large Network Analysis, *Connections*, **21 (2)**, 47–57.

[5] Beebe, N.H.F. (2002) *Nelson H.F. Beebe's Bibliographies Page*, `http://www.math.utah.edu/~beebe/ bibliographies.html`.

[6] Jones, B., *Computational Geometry Database*, February 2002, `ftp://ftp.cs.usask.ca/pub/geometry/`, `http://compgeom.cs.uiuc.edu/~jeffe/compgeom/ biblios.html`.

[7] Batagelj, V. and Zaveršnik, M. (2002) Generalized cores, submitted.

[8] Batagelj, V., Mrvar, A. and Zaveršnik, M. (1999) Partitioning approach to visualization of large graphs, In Kratochvíl, J. (ed), *Lecture notes in computer science*, 1731, Springer, Berlin, 90–97.

[9] Batagelj, V. and Mrvar, A. (2000) Some Analyses of Erdős Collaboration Graph, *Social Networks*, **22**, 173–186.