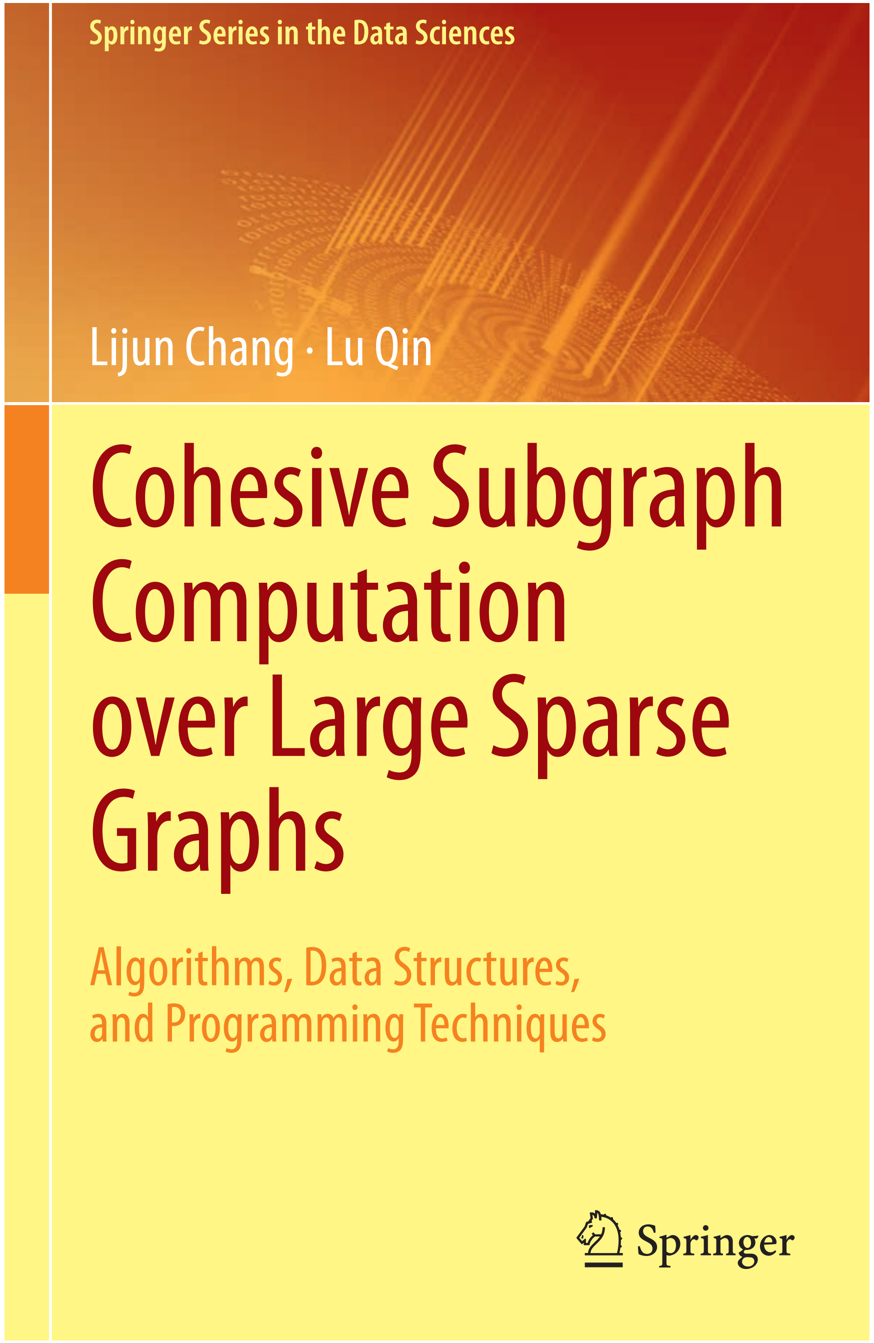
Cohesive Subgraph Computation over Large Sparse Graphs

Lijun Chang
lijun.chang@sydney.edu.au
THE UNIVERSITY OF SYDNEY

Lu Qin
lu.qin@uts.edu.au
UTS
UNIVERSITY OF TECHNOLOGY SYDNEY

Slides: lijunchang.github.io/icde19_tutorial.pdf

ICDE
MACAU 2019



Springer Series in the Data Sciences

Lijun Chang · Lu Qin

Cohesive Subgraph Computation over Large Sparse Graphs

Algorithms, Data Structures, and Programming Techniques

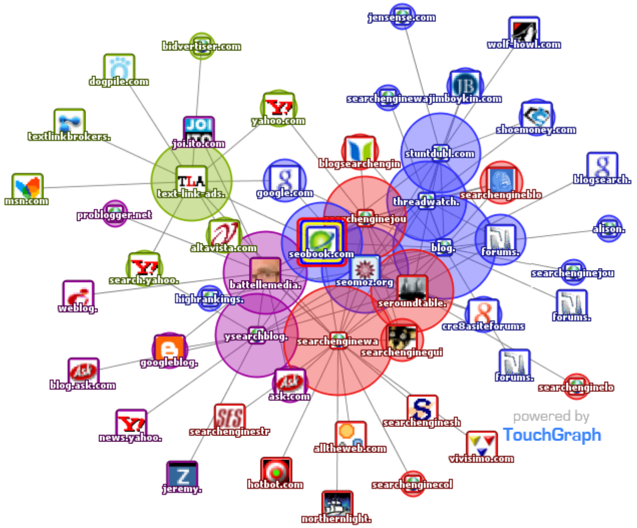
Springer

Outline

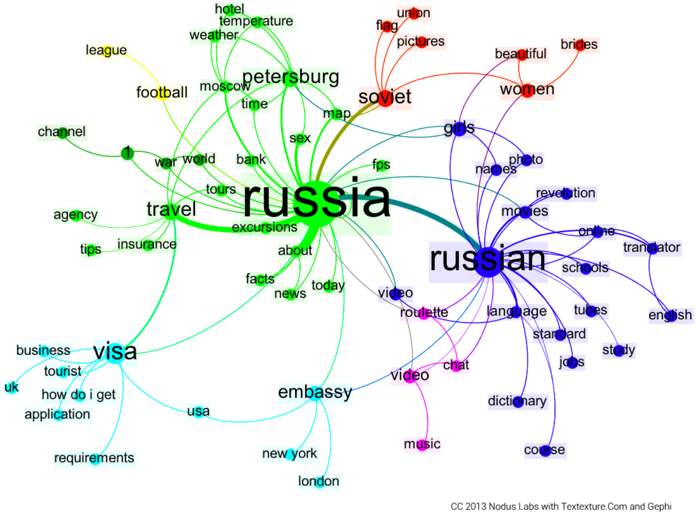
- Background
- Core Decomposition
- Densest Subgraph Computation
- Higher-order Dense Subgraph Computation
- Future Directions

Graphs are Everywhere

Social networks



Web graphs

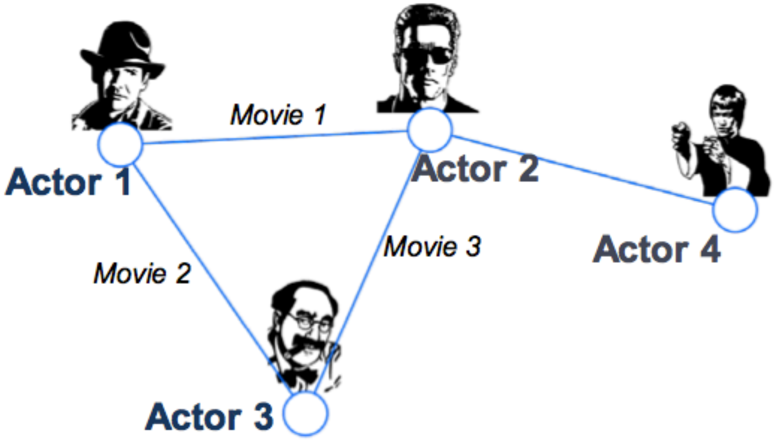


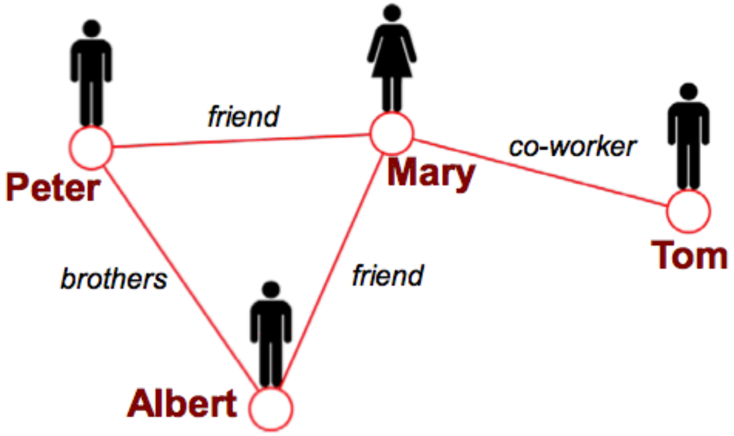
Graph of texts

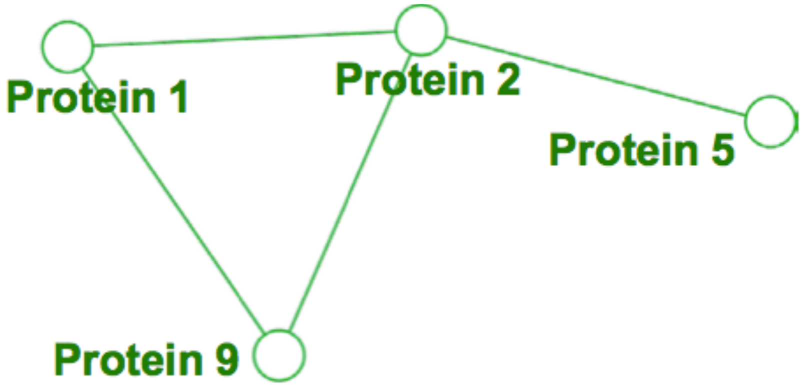
Internet of things

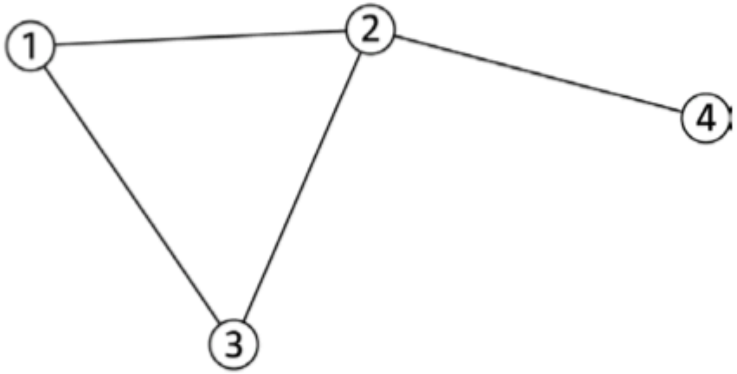
Graph Model is Simple

- A graph *G(V, E)* consists of a set of vertices *V* and a set of edges *E*





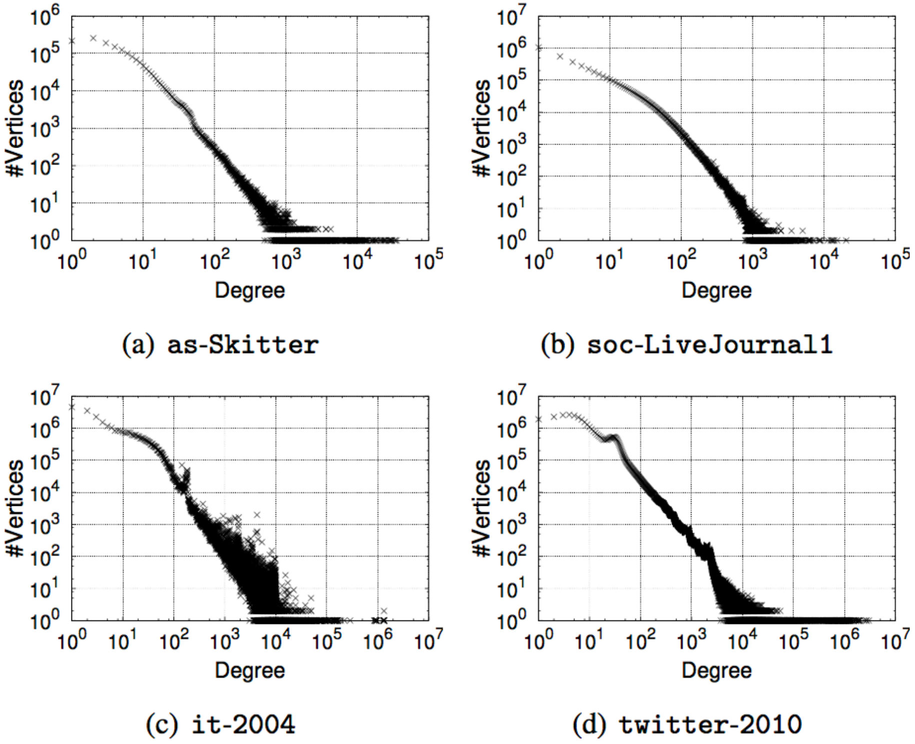




We are interested in analyzing the topological structure of real graphs!

Real Graphs are not Random Graphs

- Real graphs are not random graphs (e.g., the Erdos-Renyi random graph model), but have fascinating patterns and properties.
  - The degree distribution is skewed, following a power-law



(a) as-Skitter

(b) soc-LiveJournal1

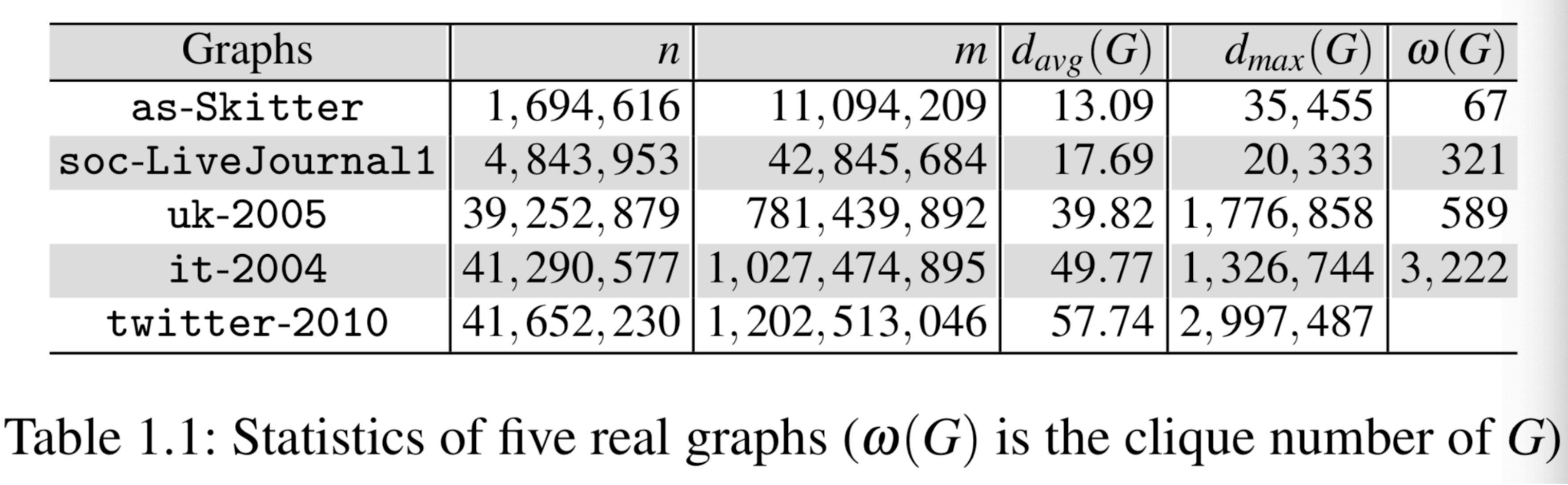
(c) it-2004

(d) twitter-2010

Fig. 1.2: Degree distributions

Real Graphs are not Random Graphs

- Real graphs are not random graphs (e.g., the Erdos-Renyi random graph model), but have fascinating patterns and properties.
  - Real graphs are globally sparse but locally dense
    - The entire graph is sparse, but there are groups of vertices with high concentration of edges within them



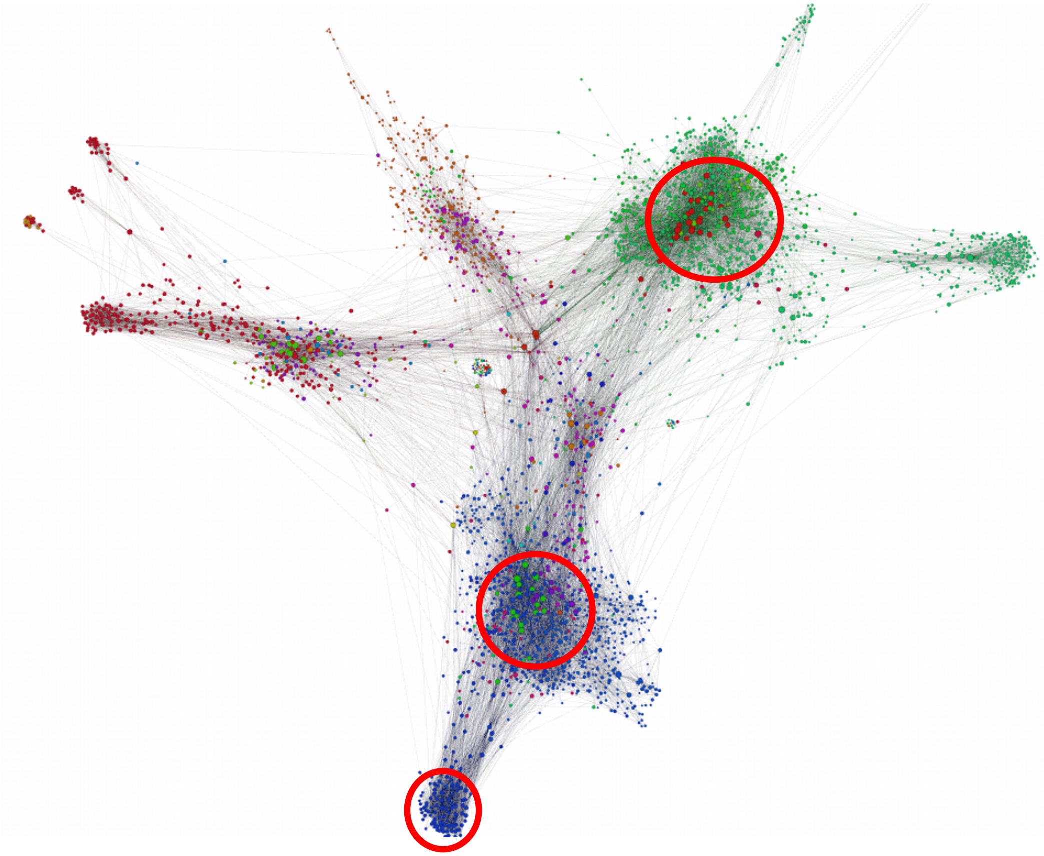
Graphs	n	m	$d_{avg}(G)$	$d_{max}(G)$	$\omega(G)$
`as-Skitter`	1,694,616	11,094,209	13.09	35,455	67
`soc-LiveJournal1`	4,843,953	42,845,684	17.69	20,333	321
`uk-2005`	39,252,879	781,439,892	39.82	1,776,858	589
`it-2004`	41,290,577	1,027,474,895	49.77	1,326,744	3,222
`twitter-2010`	41,652,230	1,202,513,046	57.74	2,997,487	

Table 1.1: Statistics of five real graphs ($\omega(G)$ is the clique number of G)

We are interested in finding "dense" subgraphs from large real graphs!

Informal Problem Definition

- Given a large sparse graph (e.g., social network, communication network, information network, biological network), find subgraphs that are densely connected or build a hierarchical structure for all dense subgraphs.



Applications of Finding Dense Subgraphs

- It has applications in any context that information can be encoded as a graph

- For example, dense subgraphs correspond to
  - Communities in *social networks*
  - Groups of web pages dealing with the same or related topics in *World Wide Web*
  - Groups of proteins having the same specific function within the cell in *biology*
  - Functional modules such as cycles and pathways in *metabolic networks*
  - Compartments in *food webs*
  - Stories in *twitter data*
  - ……

Focus of this Tutorial

- In this tutorial, we mainly focus on the fundamental technical developments of efficient dense subgraph computation
  - Efficiency is an important issue when analysing large graphs

Where to Find Large Real Graphs?

- Stanford Network Analysis Project (SNAP) [Leskovec and Krevl 2014]
  - From medium to large graphs. It includes social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks.
  - com-Friendster: 65 million vertices, 1.8 billion edges.
- Laboratory for Web Algorithmics (LAW) [Boldi and Vigna 2004]
  - Large graphs with size up to 1 billion vertices and tens of billions of edges. The networks are mainly web graphs and social networks.
  - eu-2015: 1 billion vertices, 91 billion edges.
- Network Repository [Rossi and Ahmed 2015]
  - Thousands of graphs with up to billions of vertices and tens of billions of edges.

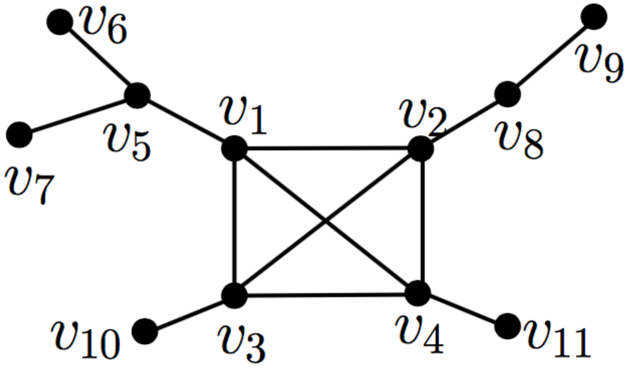
How to Store Large Sparse Graph in Memory?

- Graph representation
  - Adjacency Matrix ✗
    - Cannot store graph with over 10^5 vertices
  - Adjacency Lists ✗
    - Better, but requires *$4m$* integers
  - Adjacency Array or Compressed Sparse Row (**CSR**) ✓
    - Represents an undirected graph by *$2m+n+O(1)$* integers

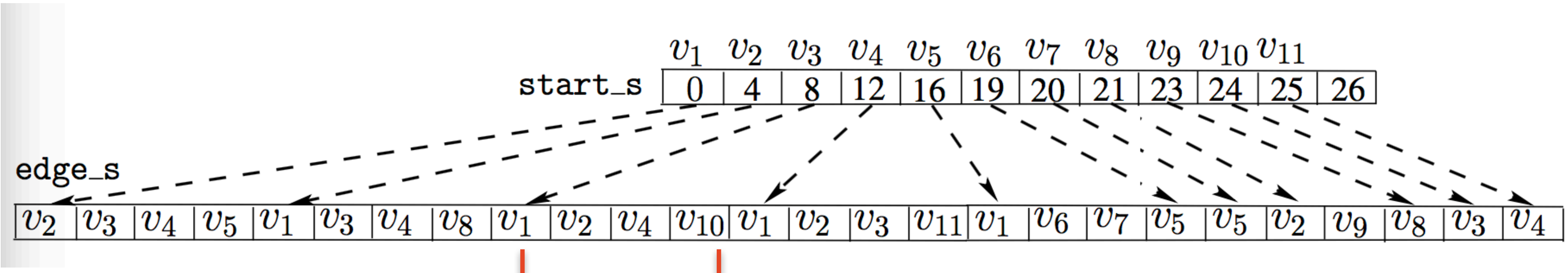
n: the number of vertices
m: the number of undirected edges

The Adjacency Array (CSR) Representation

- An example graph



- Its adjacency array representation



start_s

v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	
0	4	8	12	16	19	20	21	23	24	25	26

edge_s

v_2	v_3	v_4	v_5	v_1	v_3	v_4	v_8	v_1	v_2	v_4	v_{10}	v_1	v_2	v_3	v_{11}	v_1	v_6	v_7	v_5	v_5	v_2	v_9	v_8	v_3	v_4

Neighbors of v_3

Cohesive/Dense Subgraph Computation

- Given a graph $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$, we aim to efficiently compute dense subgraphs in G.
  - Either compute the subgraph with the highest density, or compute all (maximal) subgraphs whose density are larger than a threshold (e.g., k)
  - $n = |V|$
  - $m = |E|$

- How to measure the density of a (sub)-graph?
  - Edge ratio (2m/(n(n-1))): ratio of the number of edges to the maximum possible number of edges
    - However, small graphs usually have higher edge ratio. E.g., triangle
  - Average degree (2m/n)
  - Minimum degree

Cohesiveness Measures

- Minimum degree: core decomposition
  - Minimum number of edges each vertex participates in

- Average degree: densest subgraph
  - Average number of edges each vertex participates in

- Higher order
  - Minimum number of triangles each edge participates in: truss decomposition
  - Average number of k-cliques each vertex participates in: k-clique densest subgraph

- Edge connectivity
- …

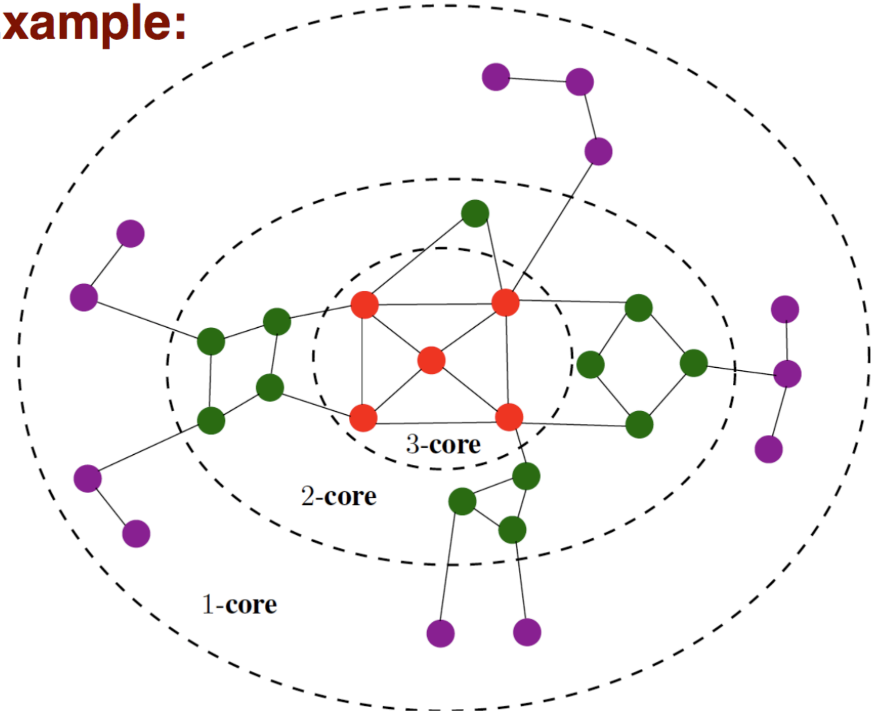
Outline

- Background
- Core Decomposition
- Densest Subgraph Computation
- Higher-order Dense Subgraph Computation
- Future Directions

Core Decomposition

- *k*-core: the maximal subgraph in which every vertex has degree at least *k* within the subgraph
- Core number *core(u)* of a vertex: the largest *k* for which the *k*-core contains the vertex

Example:



$G_0 = G$

$G_1 = \text{1-core of } G$

$G_2 = \text{2-core of } G$

$G_3 = \text{3-core of } G$

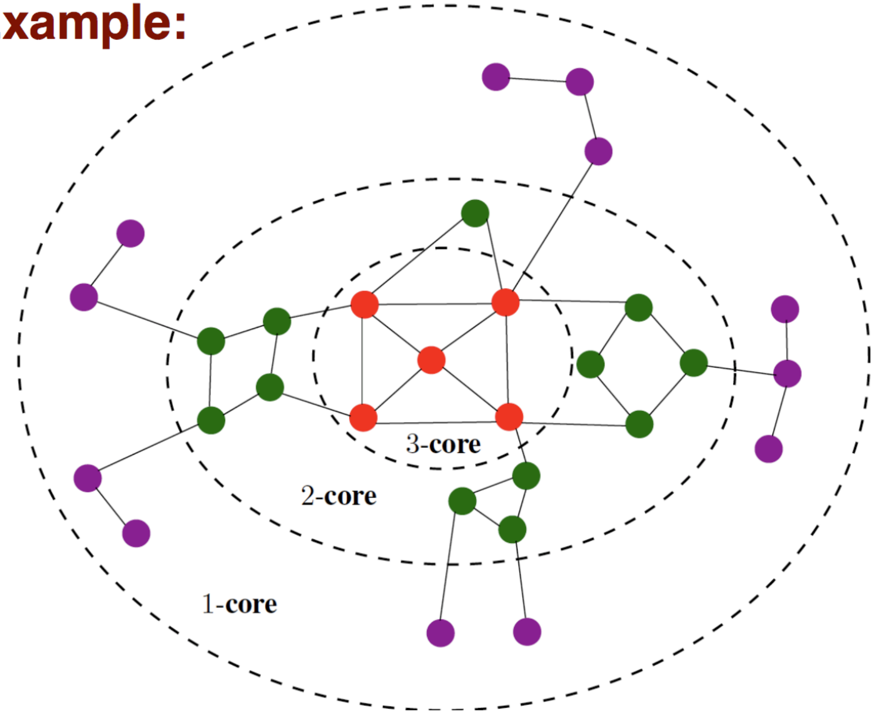
$G_0 \supseteq G_1 \supseteq G_2 \supseteq G_3$

[Malliaros et al 2016]

Core Decomposition

- Core decomposition: compute the core numbers of all vertices
  - *k*-core is the subgraph induced by all vertices with core numbers at least *k*

Example:



🟣 Core number $c_i = 1$

🟢 Core number $c_i = 2$

🔴 Core number $c_i = 3$

$G_0 = G$

$G_1 = 1\text{-core of } G$

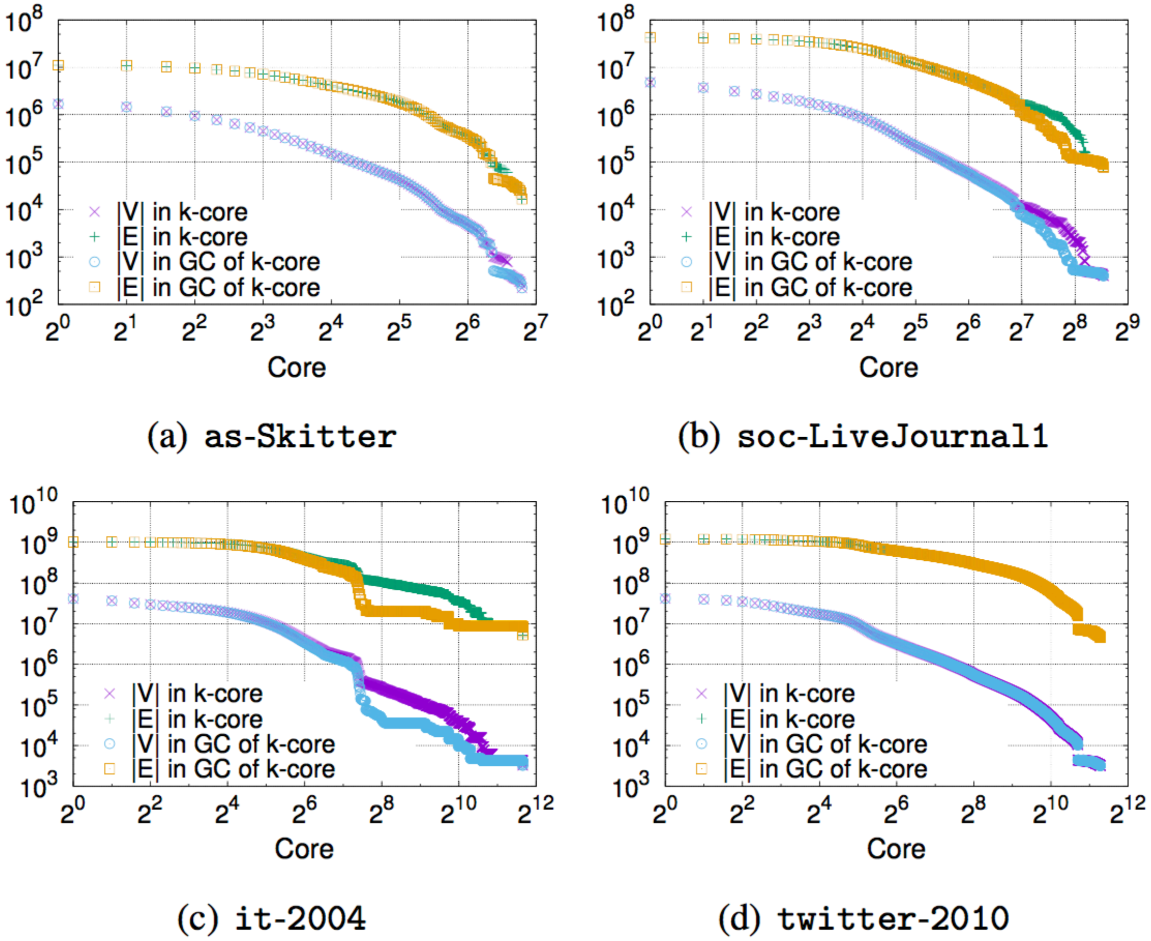
$G_2 = 2\text{-core of } G$

$G_3 = 3\text{-core of } G$

$G_0 \supseteq G_1 \supseteq G_2 \supseteq G_3$

[Malliaros et al 2016]

K-core size Distribution



(a) as-Skitter

(b) soc-LiveJournal1

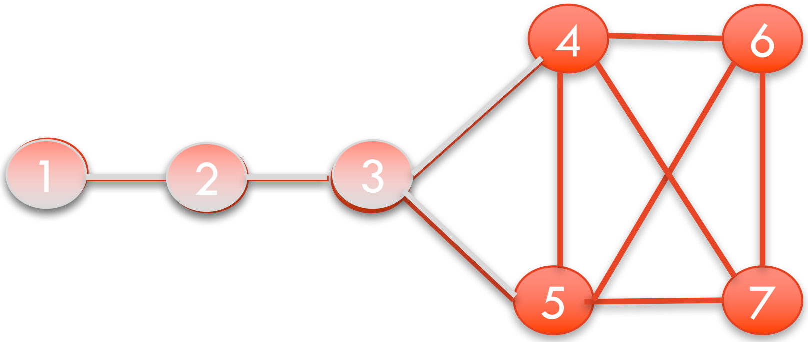
(c) it-2004

(d) twitter-2010

Fig. 3.2: Number of vertices and edges in (Giant Component of) *k*-core (varying *k*)

The Peeling Algorithm

- Basic idea for computing *k*-core: iteratively remove all vertices whose degree are smaller than *k*.
  - Core decomposition: iterate the process for *k* values increasing from *1*



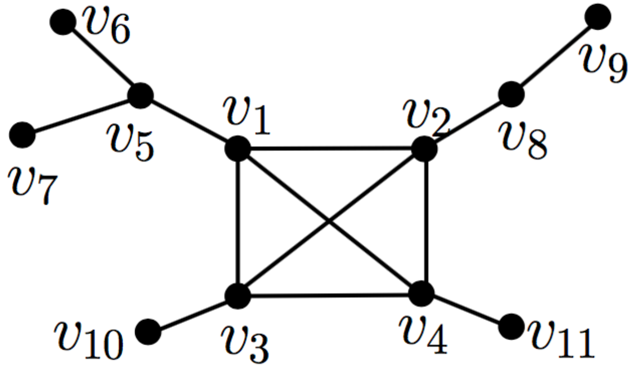
3-core

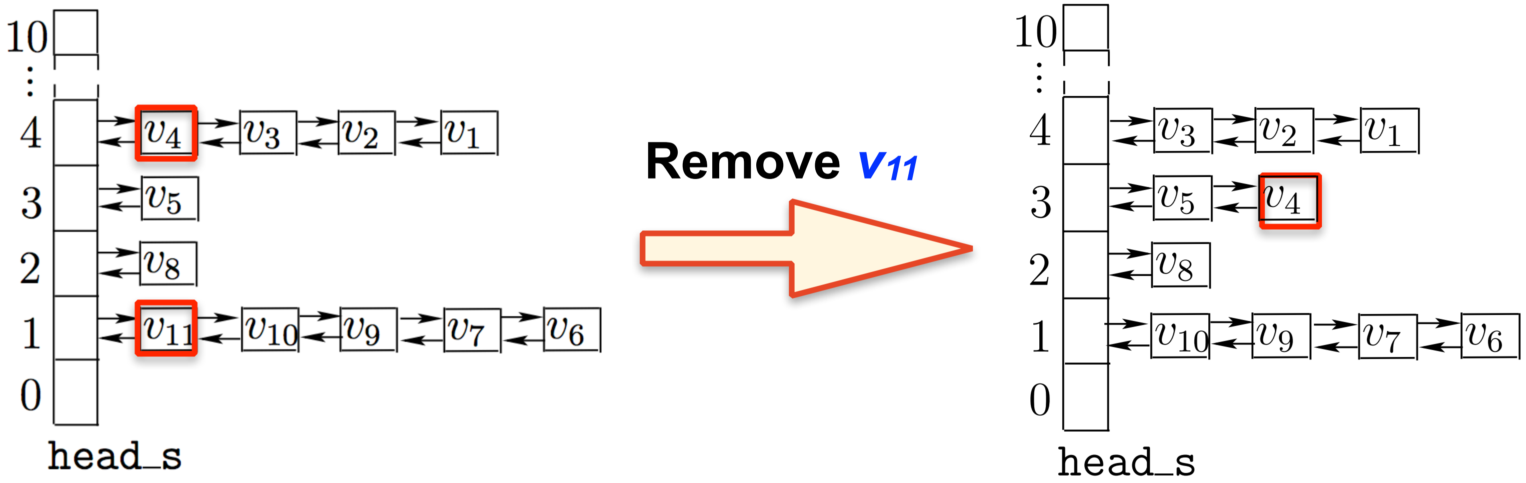
2-core

- Naively going though all vertices to find a vertex of degree smaller than *k* in each iteration will result in $O(n^2)$ time algorithm

A Linear-time Implementation

- Using a data structure to dynamically maintain the vertices of a specific degree, results in O(*m*) algorithm





Remove v_{11}

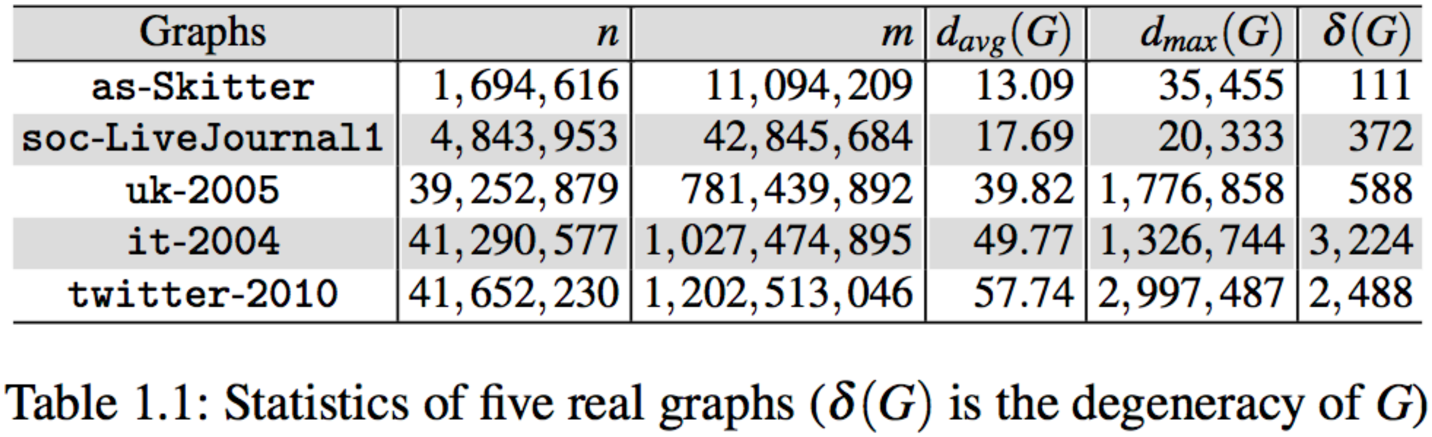
The Peeling Algorithm

- To compute k-core, we can remove an arbitrary vertex among all vertices of degree smaller than k.

- In practice, the peeling algorithm usually refers to the algorithm that iteratively removes the vertex with the smallest degree.
    - The previous data structure still can implement this algorithm to run in $O(m)$ time.

Other Applications of the Peeling Algorithm

- It computes the degeneracy $\delta(G)$ of a graph G
  - $\delta(G)$ is the maximum value among the minimum vertex degrees of all subgraphs of G
    - Each subgraph of G has a vertex with small degree (i.e., $\leq \delta(G)$)
    - There exists a subgraph with minimum degree $\delta(G)$
  - $\delta(G)$ equals the largest core number in core decomposition
  - $\delta(G) \leq \lceil \sqrt{2m+n} \rceil$
  - $\delta(G)$ measures how sparse a graph is

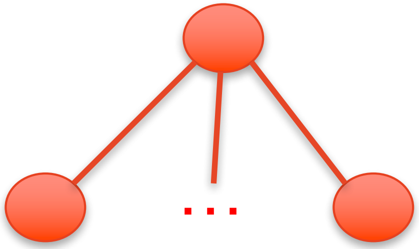


Graphs	n	m	$d_{avg}(G)$	$d_{max}(G)$	$\delta(G)$
as-Skitter	1,694,616	11,094,209	13.09	35,455	111
soc-LiveJournal1	4,843,953	42,845,684	17.69	20,333	372
uk-2005	39,252,879	781,439,892	39.82	1,776,858	588
it-2004	41,290,577	1,027,474,895	49.77	1,326,744	3,224
twitter-2010	41,652,230	1,202,513,046	57.74	2,997,487	2,488

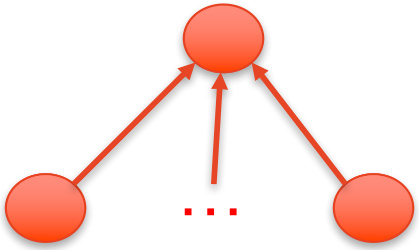
Table 1.1: Statistics of five real graphs ($\delta(G)$ is the degeneracy of G)

Other Applications of the Peeling Algorithm

- It computes the degeneracy $\delta(G)$ of a graph G
- It computes a degeneracy ordering of vertices of G
  - A permutation $(v_1, v_2, \ldots, v_n)$ of all vertices of G is a degeneracy ordering of G if every vertex v_i has the minimum degree in the subgraph induced by $\{v_i,\ldots,v_n\}$.
  - If we orient the graph according to a degeneracy ordering, then the maximum out-degree of the resulting directed graph is $\delta(G)$



Maximum degree: *n-1*



Maximum out-degree: *1*

Other Applications of the Peeling Algorithm

- It computes the degeneracy $\delta(G)$ of a graph G
- It computes a degeneracy ordering of vertices of G
- It computes an approximate value for the arboricity $\alpha(G)$ of a graph G
  - $\alpha(G)$ is the minimum number of forests needed to cover all edges of a graph
  - $\alpha(G)$ is frequently used in analyzing time complexities of algorithms, especially triangle enumeration/counting related algorithms
  - Degeneracy $\delta(G)$ tightly bounds the arboricity $\alpha(G)$ of a graph: $\alpha(G) \leq \delta(G) < 2\times\alpha(G)$

Other Applications of the Peeling Algorithm

- It computes the degeneracy $\delta(G)$ of a graph G
- It computes a degeneracy ordering of vertices of G
- It computes an approximate value for the arboricity $\alpha(G)$ of a graph G
- It computes an approximate solution to the densest subgraph (will be covered later)

H-index-based Local Algorithm

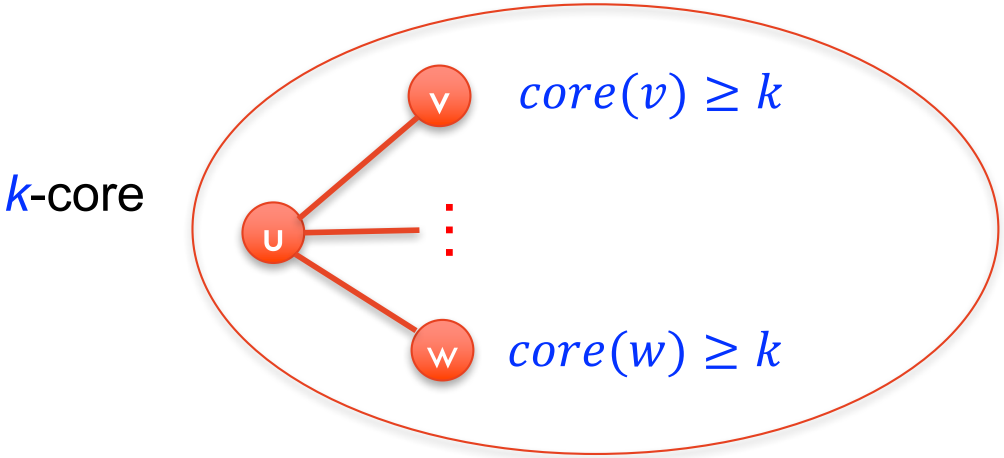
- The peeling algorithm is inherently sequential, and has limited parallelizability.

- There is an H-index-based local algorithm that works well in practice for different settings
  - e.g., parallel setting, distributed setting, I/O-efficient setting, in-memory

- Given a multi-set *S* of positive numbers, *h-index(S)* is the largest integer *k* such that $|\{s \in S : s \geq k\}| \geq k$
  - E.g. h-index({1,1,1,1}) = 1
  - h-index({4,3,2,1}) = 2

H-index-based Local Algorithm

- Fact 1: let $C_u = \{core(v): v \in N(u)\}$, then $core(u) = h{-}index(C_u)$
  - Let $\mathrm{k} = core(u)$, u must have at least *k* neighbors in the *k*-core.



- Fact 2: let $\overline{core}(v)$ be an upper bound of $core(v)$ and $\bar{C}_u = \{\overline{core}(v): v \in N(u)\}$, then $h{-}index(\bar{C}_u)$ is an upper bound of $core(u)$

H-index-based Local Algorithm

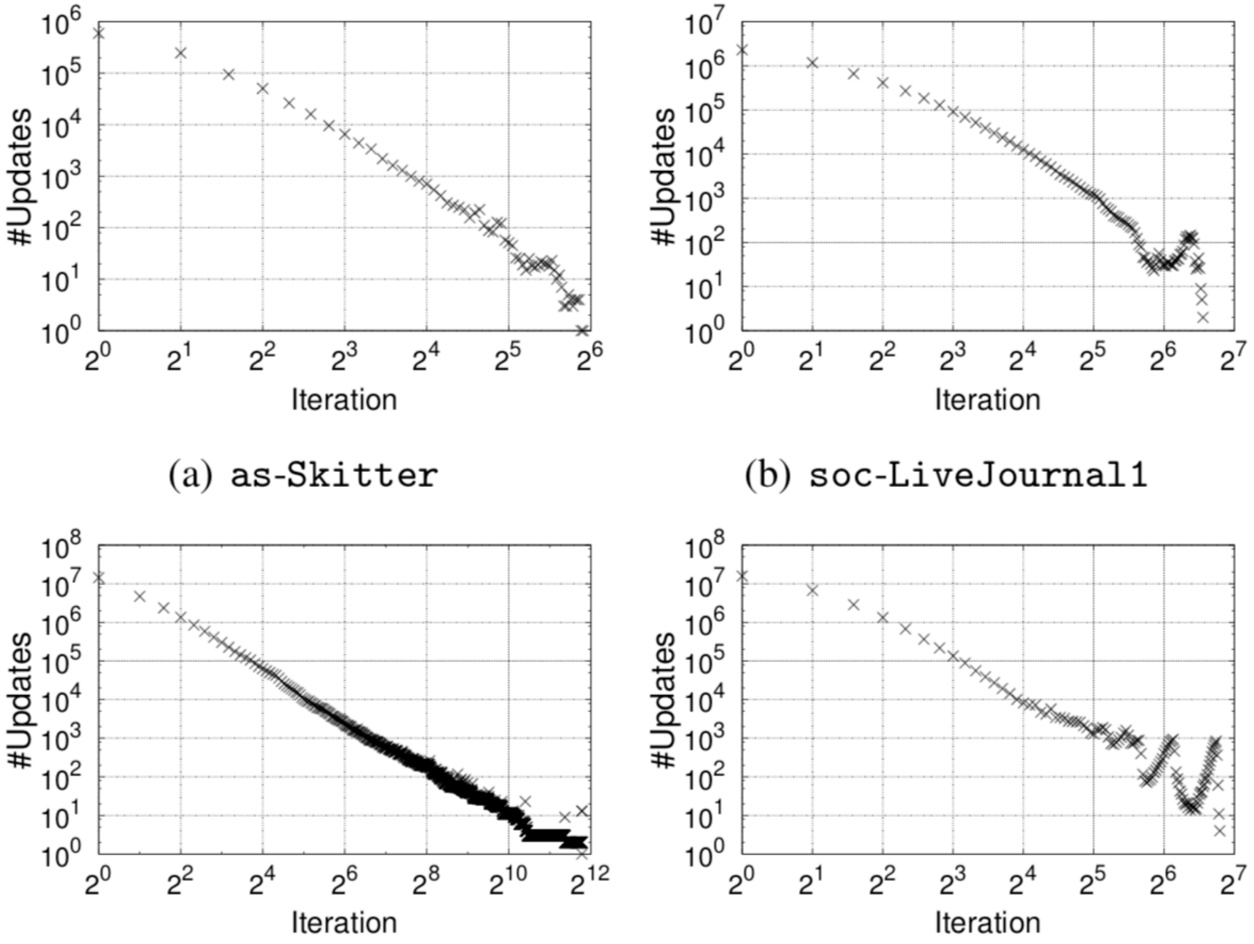
- Fact 1: let $C_u = \{core(v): v \in N(u)\}$, then $core(u) = h\text{−}index(C_u)$
- Fact 2: let $\bar{C}_u = \{\overline{core}(v): v \in N(u)\}$, then $core(u) \leq h\text{−}index(\bar{C}_u)$

- Algorithm:
  - Initialize $\overline{core}(v)$ to be the degree of v for all vertices
  - Repeat until converge: reassign $\overline{core}(u)$ as $h\text{−}index(\bar{C}_u)$ for all vertices

- h-index is monotone
  - The upper bounds cannot increase
  - The upper bounds converge to the true core numbers.

- Optimization: do not need to update the upper bound for every vertex in each iteration

H-index-based Local Algorithm



(a) as-Skitter

(b) soc-LiveJournal1

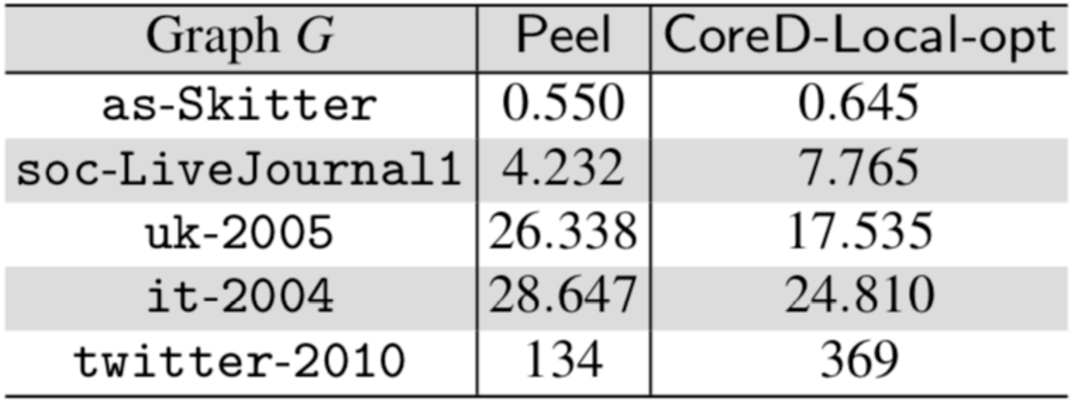
(c) it-2004

(d) twitter-2010

Fig. 3.5: Number of updates in different iterations

H-index-based Local Algorithm

- Empirical in-memory running time comparison (in seconds)



Graph G	Peel	CoreD-Local-opt
as-Skitter	0.550	0.645
soc-LiveJournal1	4.232	7.765
uk-2005	26.338	17.535
it-2004	28.647	24.810
twitter-2010	134	369

- The running time highly depends on the processing order of vertices
  - E.g., if processing vertices in the degeneracy ordering, then the time complexity is linear

Other Works on Core Decomposition

- Core decomposition for dynamic graph
  - How to maintain the core number when graph changes?
  - [Zhang et al 2017]

- Core decomposition for uncertain graph
  - [Bonchi et al 2014]

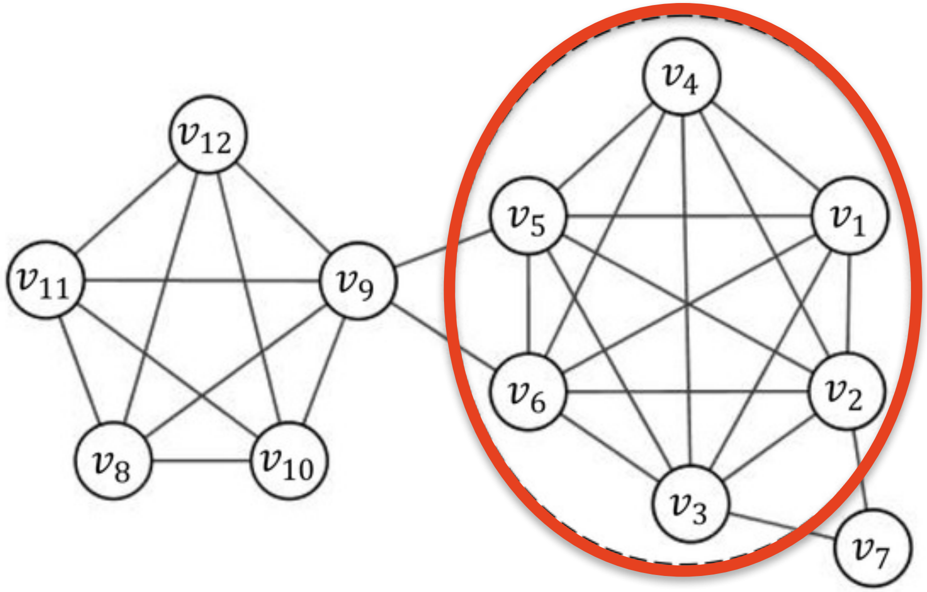
- Core decomposition for directed graph
  - [Giatsidis et al 2011]

Outline

- Background
- Core Decomposition
- Densest Subgraph Computation
- Higher-order Dense Subgraph Computation
- Future Directions

Densest Subgraph

- Find the subset S of vertices in G where the induced subgraph of G by S has the largest average degree among all subsets



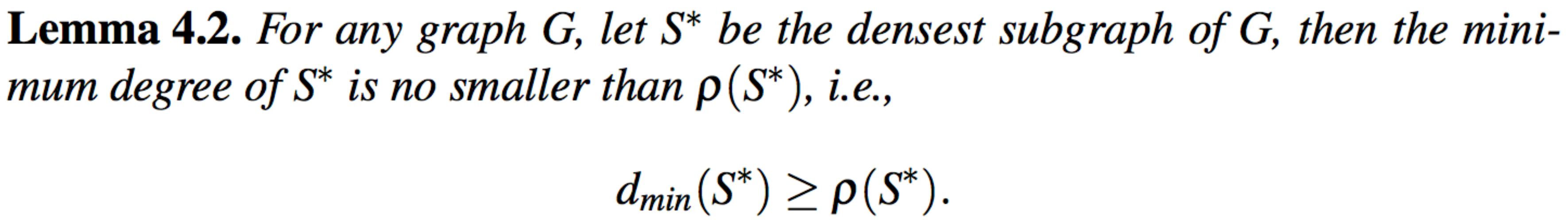
Greedy-based Approximation Algorithm

- Iteratively remove the vertex with the minimum degree from the graph (This is the same as the previous peeling algorithm)
  - We obtain n subgraphs during the process
  - Return the one with the maximum average degree as the result

- Time complexity: O(m)

Greedy-based Approximation Algorithm

- Approximation ratio
  - Density $\rho(S)$ of S: total number of edges divided by total number of vertices (half of the average degree)
  - Upper bound of the maximum density $\rho(S^*)$



Lemma 4.2. *For any graph G, let S^* be the densest subgraph of G, then the minimum degree of S^* is no smaller than $\rho(S^*)$, i.e.,*

$$d_{min}(S^*) \geq \rho(S^*).$$

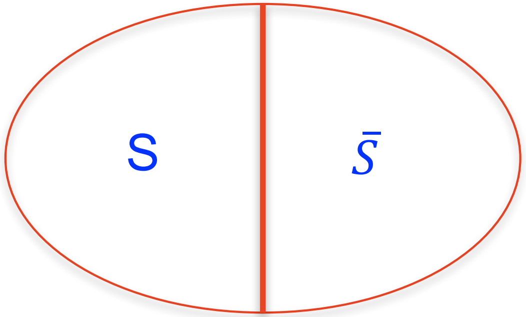
- $\rho(S^*) \leq d_{min}(S^*) \leq \delta(G)$: recall $\delta(G)$ is the maximum value among the minimum vertex degrees of all subgraphs of G

  - Let's look at the $\delta(G)$-core: recall there exists a subgraph with minimum degree $\delta(G)$
    - Its density is at least $\delta(G)/2 \geq \rho(S^*)/2$
  - The approximation ratio of the greedy algorithm is $1/2$

Goldberg's Algorithm for Densest Subgraph

[Goldberg 1984]

- Decision version of the densest subgraph problem: Is there a subgraph S with density larger than λ ?
  - $\rho(S)$: density of S (half of the average degree)
  - $\bar{S}$: vertices of G not in S
  - $E(S,\bar{S})$: edges between S and $\bar{S}$

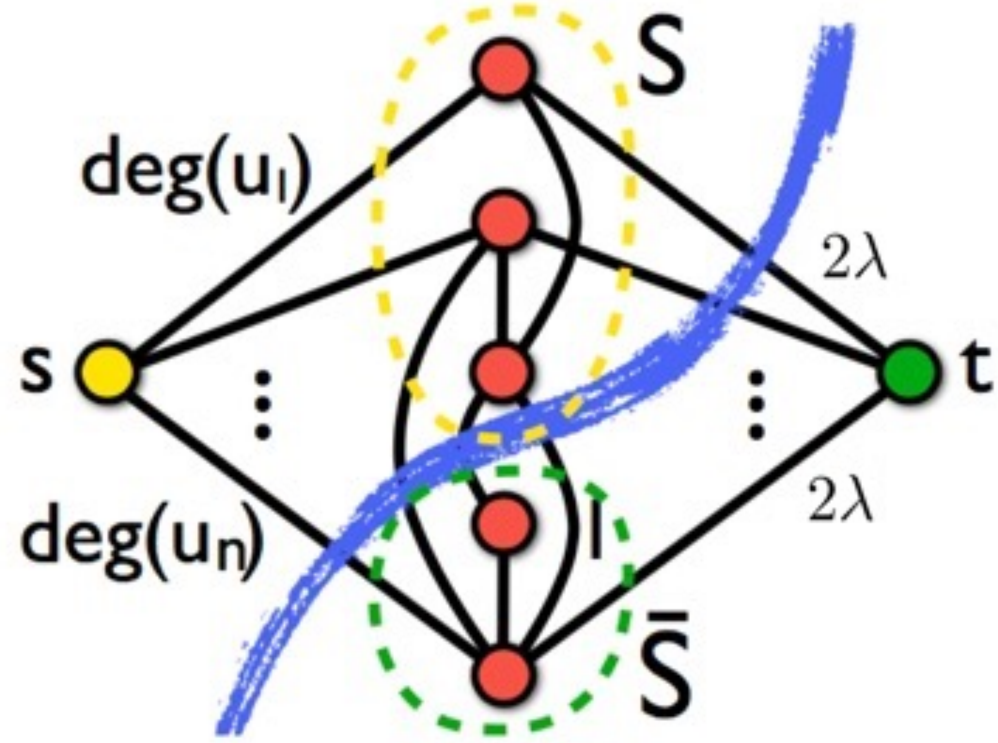


$$
\begin{aligned}
& \rho(S) > \lambda \\
\Longleftrightarrow & 2|E(S)| > 2\lambda|S| \\
\Longleftrightarrow & \sum_{u\in S} d(u) - |E(S,\bar{S})| > 2\lambda|S| \\
\Longleftrightarrow & \sum_{u\in S} d(u) + \sum_{u\in \bar{S}} d(u) - \sum_{u\in \bar{S}} d(u) - |E(S,\bar{S})| > 2\lambda|S| \\
\Longleftrightarrow & \sum_{u\in \bar{S}} d(u) + |E(S,\bar{S})| + 2\lambda|S| < 2|E|
\end{aligned}
$$

Goldberg’s Algorithm for Densest Subgraph

[Goldberg 1984]

There is S s.t. $\rho(S) > \lambda \iff \sum_{u \in \bar{S}} d(u) + |E(S, \bar{S})| + 2\lambda|S| < 2|E|$



G_λ

[Gionis and Tsourakakis 2015]

- $\lambda < \rho(S^*)$ iff the minimum cut of G_λ is of value smaller than 2m
- $\lambda \geq \rho(S^*)$ iff the minimum cut of G_λ is of value exactly 2m

Goldberg's Algorithm for Densest Subgraph

[Goldberg 1984]

- Thus, we can do binary search on λ.
  - When λ is smaller than but very close to $\rho(S^*)$, then the minimum cut of the graph G_λ corresponds to a densest subgraph of G

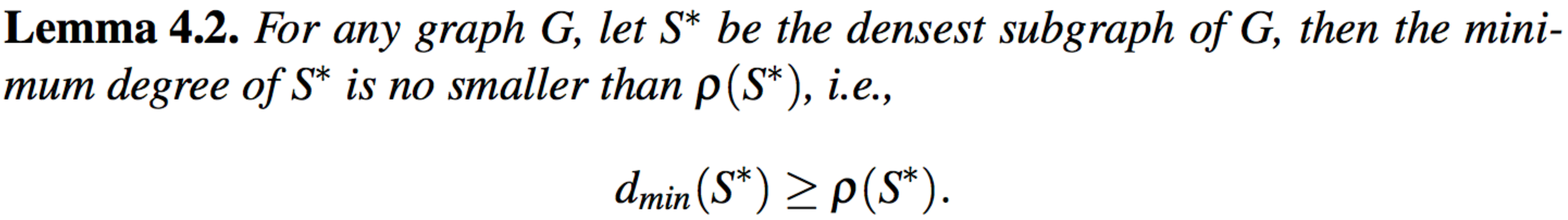
- But λ is a fractional number, when to stop?
  - For any two subgraphs S_1 and S_2 with $\rho(S_1) > \rho(S_2)$, it holds that $\rho(S_1) - \rho(S_2) \geq \frac{1}{n(n-1)}$

- Time complexity of Goldberg's algorithm
  - $O(\log n)$ minimum cut computations, each for a different λ value
  - By using parametric maximum flow techniques, can be implemented to run in $O(\mathrm{n} \cdot \mathrm{m} \cdot \log \frac{n^2}{m})$ time

Data Reduction for Densest Subgraph Computation

- Goldberg's algorithm cannot be directly applied to large graphs, due to the high time complexity

- We can reduce the graph instance for Goldberg's algorithm
  - Real-world graphs are power-law graphs, many vertices are of small degree and thus cannot be in the densest subgraph



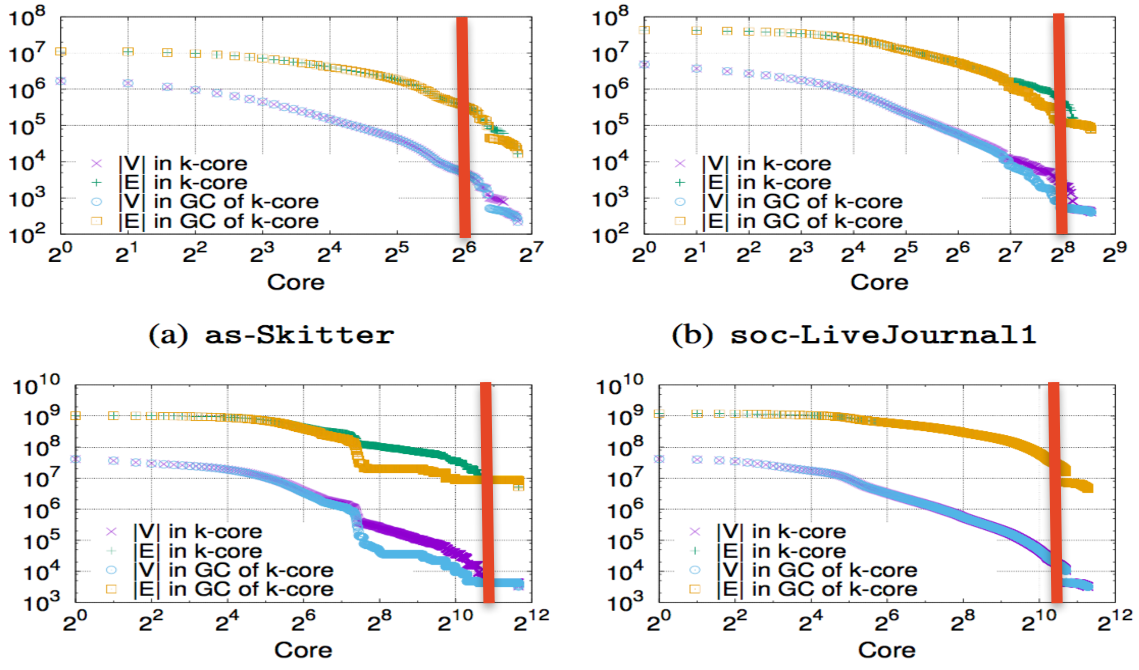
Lemma 4.2. *For any graph G, let S^* be the densest subgraph of G, then the minimum degree of S^* is no smaller than $\rho(S^*)$, i.e.,*

$$d_{min}(S^*) \geq \rho(S^*).$$

  - The density of the $\delta(G)$-core is at least $\delta(G)/2$
  - Thus, we can remove all vertices whose degree are smaller than $\delta(G)/2$

Data Reduction for Densest Subgraph Computation

- Thus, to exactly compute the densest subgraph, we only need to consider the $\delta(G)/2$-core, rather than the entire graph



(a) as-Skitter

(b) soc-LiveJournal1

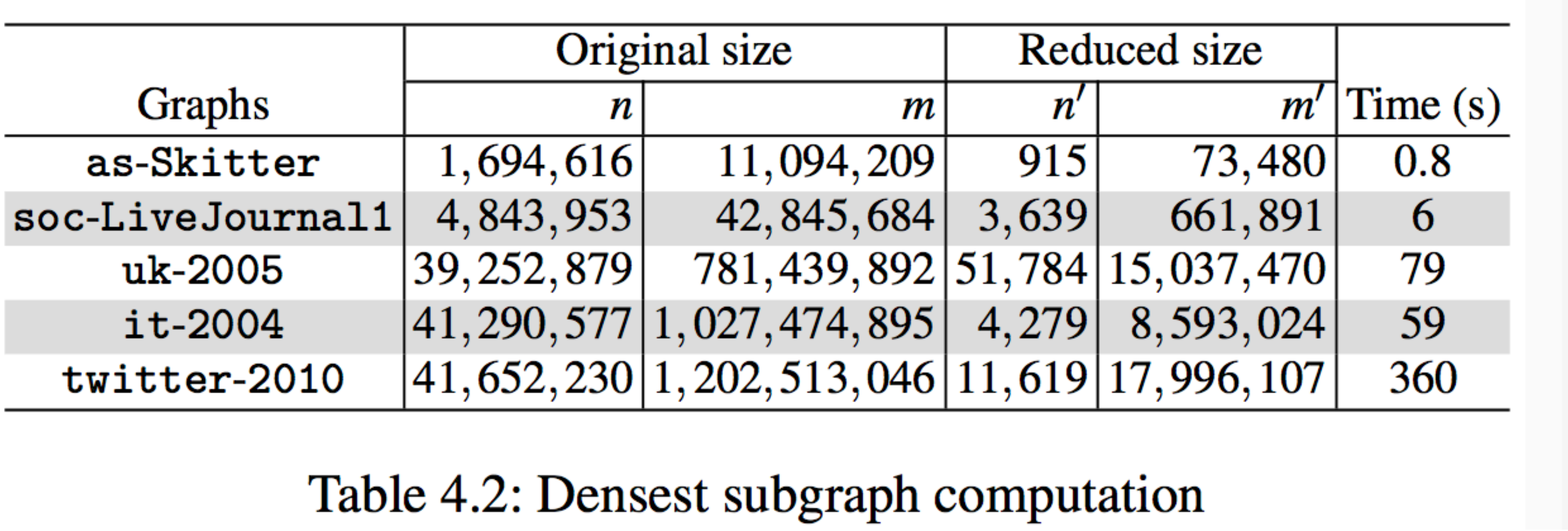
(c) it-2004

(d) twitter-2010

Fig. 3.2: Number of vertices and edges in (Giant Component of) *k*-core (varying *k*)

Data Reduction for Densest Subgraph Computation

- In practice, we can first run the greedy algorithm to get an approximate densest subgraph S, and then run Goldberg's algorithm on the $\lceil \rho(S) \rceil$-core of G



Graphs	Original size		Reduced size		Time (s)
	n	m	n'	m'	
as-Skitter	1,694,616	11,094,209	915	73,480	0.8
soc-LiveJournal1	4,843,953	42,845,684	3,639	661,891	6
uk-2005	39,252,879	781,439,892	51,784	15,037,470	79
it-2004	41,290,577	1,027,474,895	4,279	8,593,024	59
twitter-2010	41,652,230	1,202,513,046	11,619	17,996,107	360

Table 4.2: Densest subgraph computation

Other Works on Densest Subgraph Computation

- The densest subgraph can also be computed by linear programming
  - [Charikar 2000]

- Densest subgraph computation in dynamic graphs
  - [Epasto et al 2015]

- Locally densest subgraph
  - [Qin et al 2015]

- Density-friendly graph decomposition
  - [Danish et al 2017]

Outline

- Background
- Core Decomposition
- Densest Subgraph Computation
- Higher-order Dense Subgraph Computation
- Future Directions

Higher-order Structures

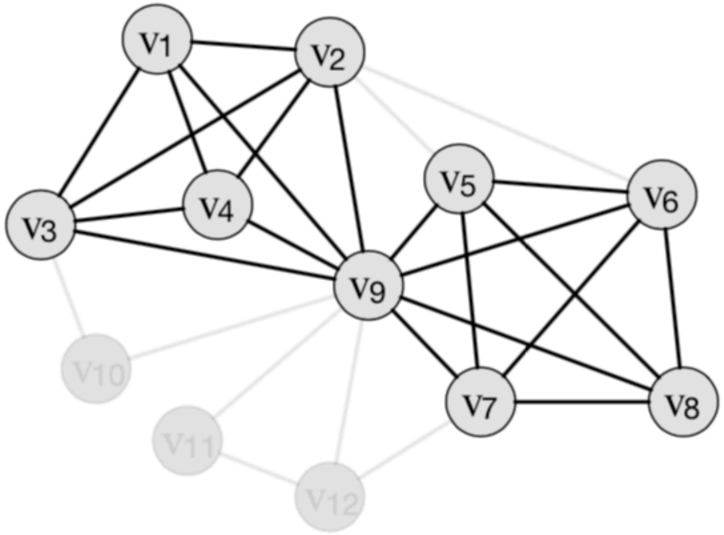
- Previous, we focused on vertices and edges

- Now, let's consider higher-order structures, k-cliques (complete graphs with k vertices), which usually will find denser subgraphs
  - A vertex is a 1-clique
  - An edge is a 2-clique
  - A triangle is a 3-clique

- Higher-order core decomposition
  - Truss decomposition
  - Nucleus decomposition
- Higher-order densest subgraph

Truss Decomposition

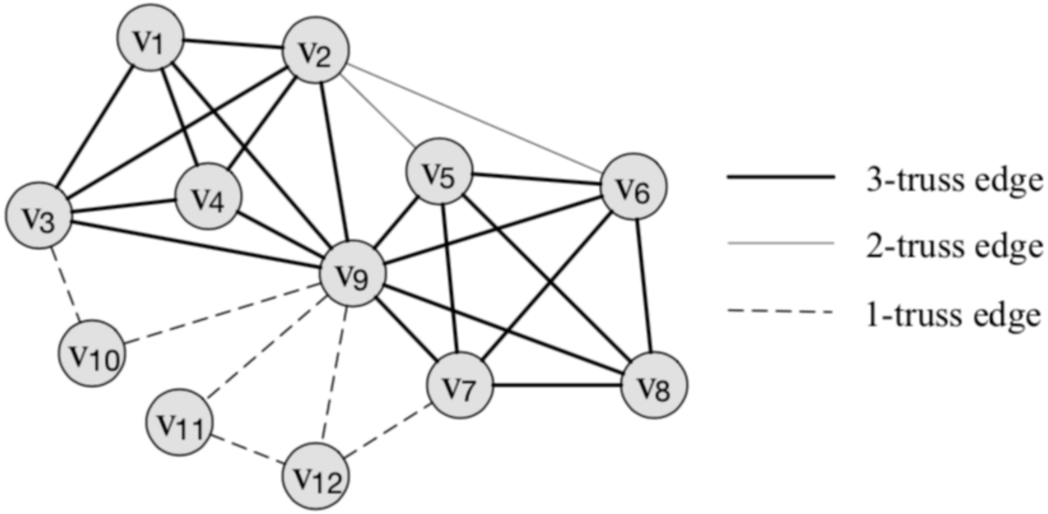
- k-truss: the *maximal* subgraph in which every edge participates in at least k triangles
- k-core: the maximal subgraph in which every vertex participates in at least k edges
- Like k-cores, k-trusses are nested



3-truss

Truss Decomposition

- Truss number *truss(u,v)* of an edge: the largest *k* for which the *k*-truss contains the edge
- Truss decomposition: compute the truss number for each edge
  - *k*-truss is the subgraph induced by all edges with truss numbers at least *k*



Truss decomposition

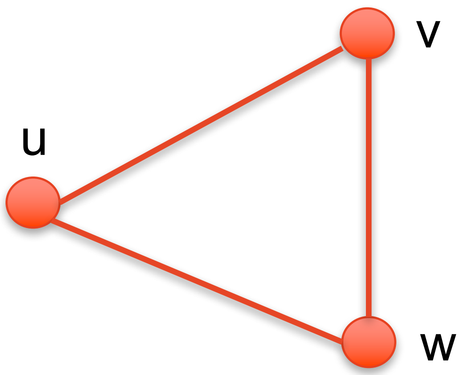
Computing Truss Decomposition

- Extend the peeling algorithm
  - Iteratively remove the edge that participates in the fewest number of triangles
  - How to efficiently compute the number of triangles for each vertex?
  - How to efficiently update the number of triangles after deleting one edge?

  - This needs an efficient triangle enumeration algorithm.

Triangle Enumeration

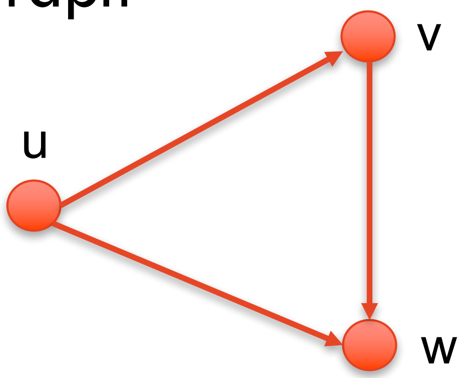
- How to efficiently enumerate all triangles in a graph?

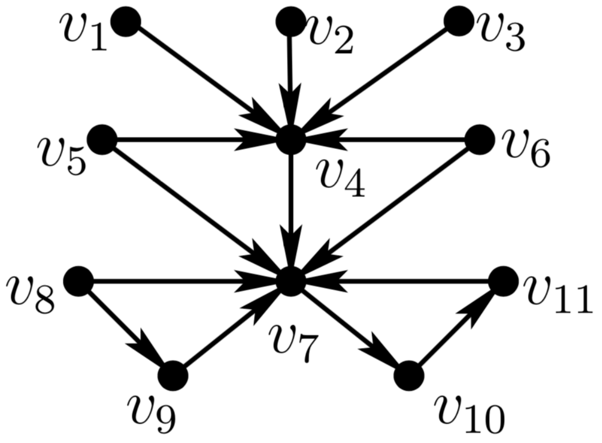


- How about enumerating wedge (v,u,w) and check the existence of edge (v,w)?
  - The time complexity will be $\sum_{u \in V}(d(u))^2$
    - This is higher than $m^{\frac{3}{2}}$, which bounds the maximum number of triangles
    - E.g., consider a star graph

Triangle Enumeration

- We can improve the time complexity by orienting the input graph



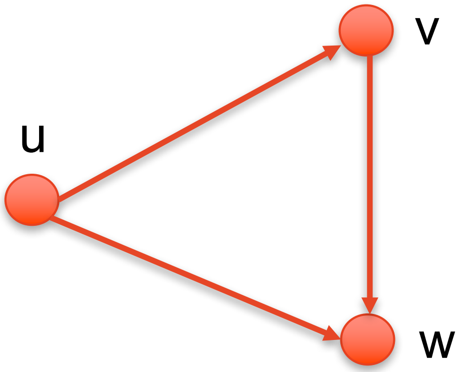


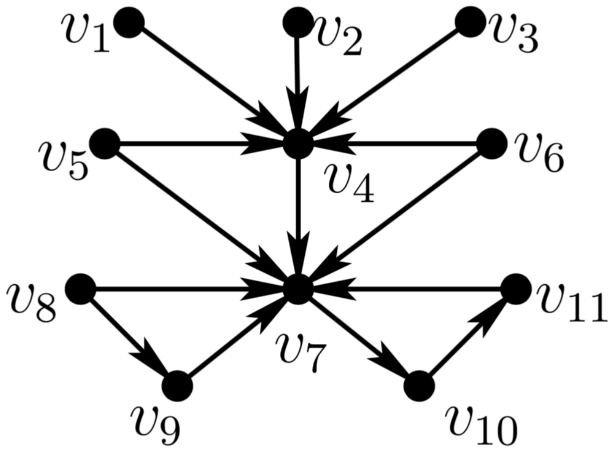
Oriented graph

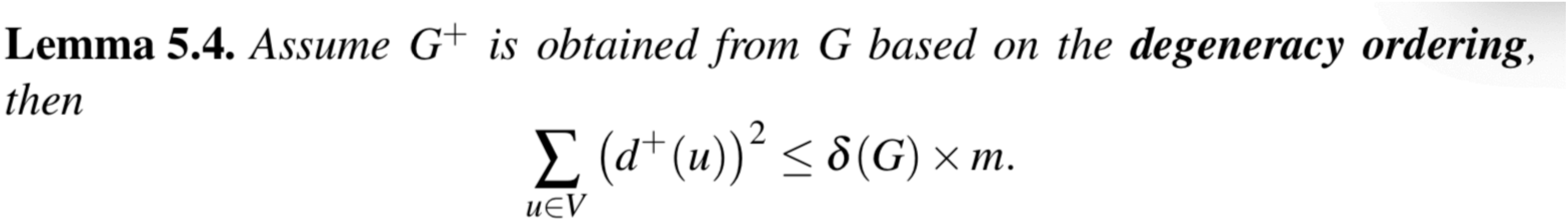
- How about enumerating wedge (v,u,w) and check the existence of edge (v,w)?
  - The time complexity will be $\sum_{u \in V} (d^+(u))^2$
  - This can be small if we orient the graph smartly

Triangle Enumeration

How about enumerating wedge (v,u,w) and check the existence of edge (v,w)?







Lemma 5.4. *Assume G^+ is obtained from G based on the* ***degeneracy ordering****, then*

$$\sum_{u \in V} \left(d^+(u)\right)^2 \leq \delta(G) \times m.$$

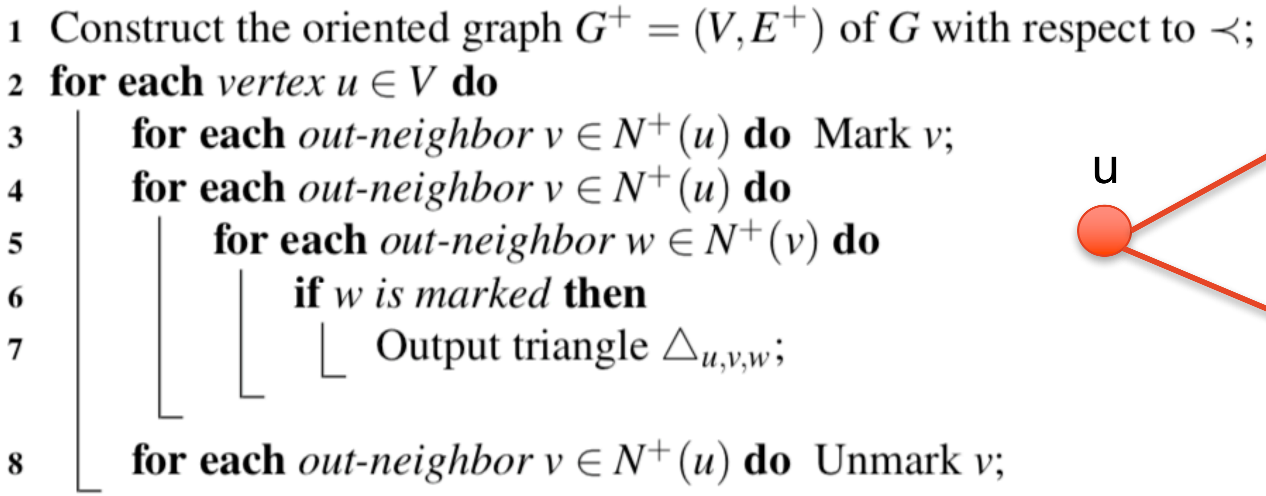
Recall that $\delta(G) \leq \lceil \sqrt{2m+n} \rceil$

The total number of wedges checked is $O(m^{\frac{3}{2}})$

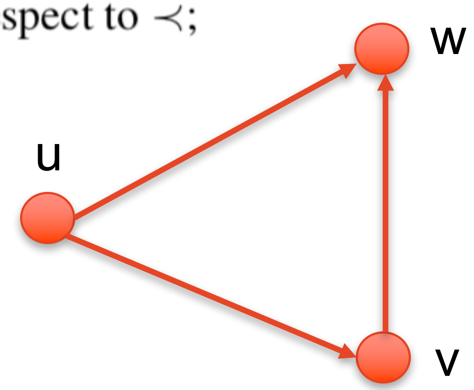
How to check the existence of an edge? Hash table!

Triangle Enumeration

- Hash table has both space and time overhead. Can we avoid the hash-table? Yes!
  - Arrange the edge-existence checking in a smart way.



```
Construct the oriented graph $G^+ = (V, E^+)$ of $G$ with respect to $\prec$;
for each vertex $u \in V$ do
    for each out-neighbor $v \in N^+(u)$ do Mark $v$;
    for each out-neighbor $v \in N^+(u)$ do
        for each out-neighbor $w \in N^+(v)$ do
            if $w$ is marked then
                Output triangle $\triangle_{u,v,w}$;
    for each out-neighbor $v \in N^+(u)$ do Unmark $v$;
```



  - This is to check whether *u* is connected to its 2-hop out-neighbor *w*
  - All such checks for the same *u* are grouped together. Hash table is not needed!
  - The time complexity is no longer $\sum_{u \in V} (d^+(u))^2$
    - but $\sum_{v \in V} (d^+(v) \times d^-(v))$

Triangle Enumeration

```
Construct the oriented graph $G^+ = (V, E^+)$ of $G$ with respect to $\prec$;
for each vertex $u \in V$ do
    for each out-neighbor $v \in N^+(u)$ do Mark $v$;
    for each out-neighbor $v \in N^+(u)$ do
        for each out-neighbor $w \in N^+(v)$ do
            if $w$ is marked then
                Output triangle $\triangle_{u,v,w}$;
    for each out-neighbor $v \in N^+(u)$ do Unmark $v$;
```

Lemma 5.5. *Assume G^+ is obtained from G based on the* ***degeneracy ordering****, then*

$$\sum_{u \in V} d^-(u) \times d^+(u) \leq (2\alpha(G) - 1) \times m.$$

$$\alpha(G) \leq \left\lceil \frac{\sqrt{2m+n}}{2} \right\rceil$$

Lemma 5.3. *Assume G^+ is obtained from G based on the* ***degree*** **decreasing** ***ordering****, then*

$$\sum_{u \in V} d^-(u) \times d^+(u) \leq 2 \times \alpha(G) \times m.$$

As computing degeneracy ordering takes a significant portion of the total time, degree decreasing or increasing ordering is used in practice.

Computing Truss Decomposition

- Extend the peeling algorithm
  - Iteratively remove the edge that participates in the fewest number of triangles
  - How to efficiently compute the number of triangles for each vertex?
    - Enumerate all triangles by the algorithm in previous slide
  - How to efficiently update the number of triangles after deleting one edge (u,v)?
    - Intersect the neighbor-sets of u and v in $\min\{\mathrm{d}(\mathrm{u}), \mathrm{d}(\mathrm{v})\}$ time

Hash table is needed here!

- The algorithm runs in $O(\alpha(G) \times m)$ time
  - $\alpha(G)$ is the arboricity of G, and is small for real graphs

$$\sum_{(u,v)\in E^+} \min\{d(u), d(v)\} \leq 2 \times \alpha(G) \times m$$

Nucleus Decomposition

triangles

- k-(r,s)-nucleus: the maximal union g of s-cliques in G such that for each r-clique C in g, there are at least k s-cliques in g containing C edges
  - k-core is a k-(1,2)-nucleus

    Iteratively remove vertices with fewer than k edges containing the vertex, the remaining edges form the *k*-core
  - k-truss is a k-(2,3)-nucleus

    Iteratively remove edges with fewer than k triangles containing the edge, the remaining triangles form the *k*-truss

Iteratively remove r-cliques with fewer than k s-cliques containing the r-clique, the remaining s-cliques form the *k*-(r,s)-nucleus?

Yes!

Nucleus Decomposition

- Iteratively remove r-cliques with fewer than k s-cliques containing the r-clique, the remaining s-cliques form the *k*-(r,s)-nucleus
- Let's consider the hyper-graph $\mathbb{G} = (\mathbb{V}, \mathbb{E})$
  - $\mathbb{V}$ is the set of r-cliques in G
  - $\mathbb{E}$ is the set of s-cliques in G (hyper-edges)
  - Each hyper-edge (s-clique) in $\mathbb{G}$ connects to all r-cliques contained in the s-clique
  - In truss decomposition, r = 2 and s = 3

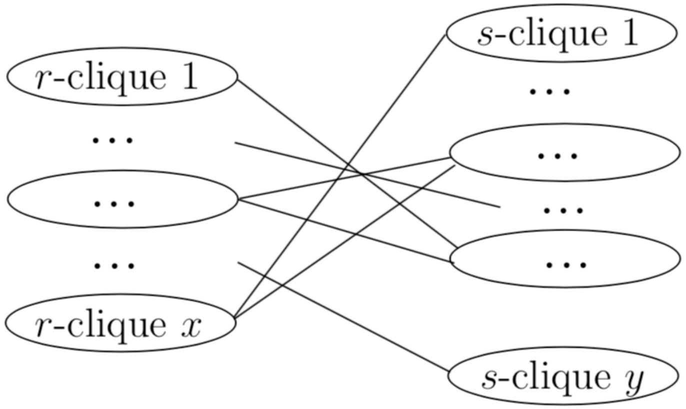


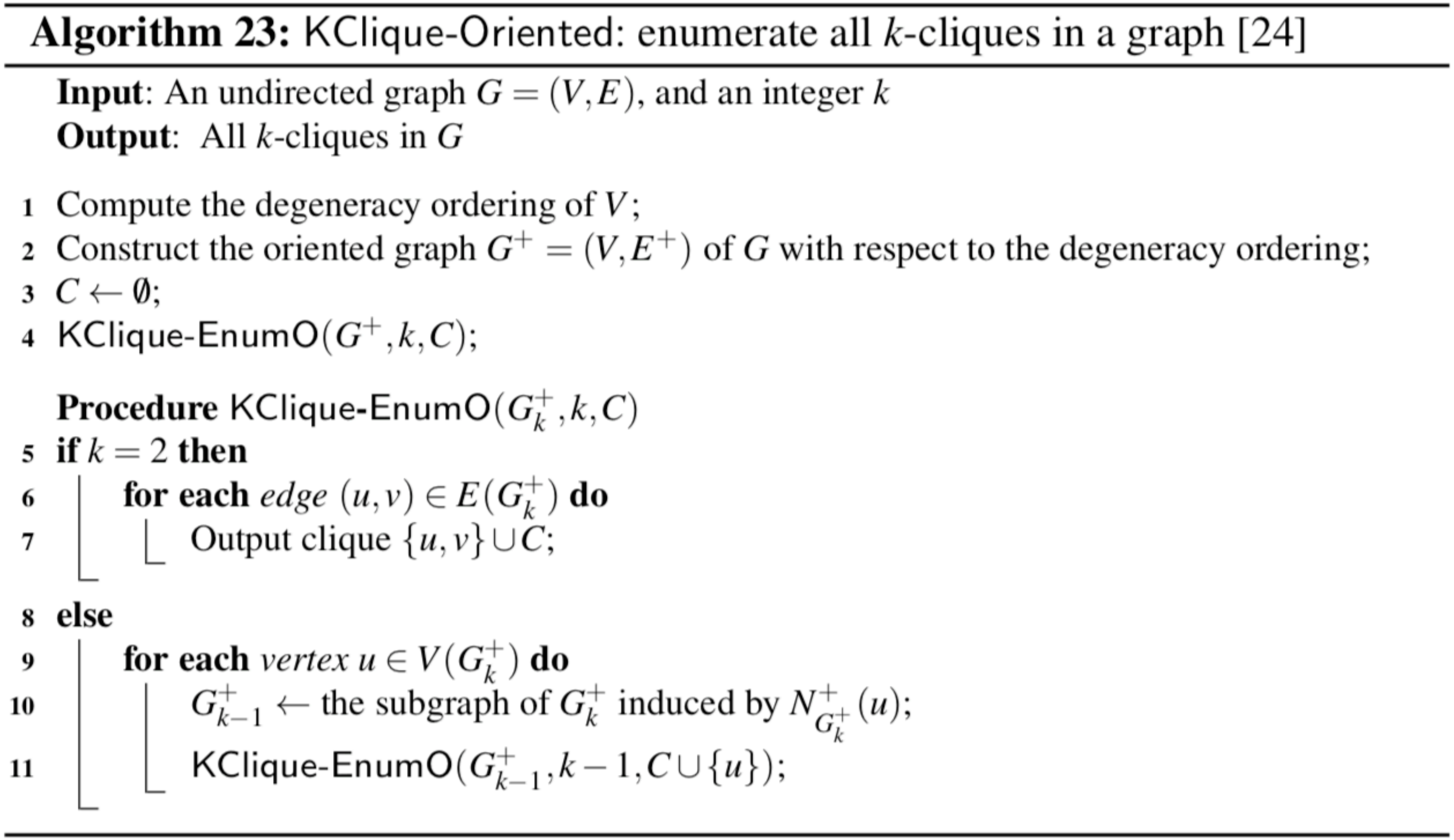
Fig. 5.4: Bipartite graph for nucleus decomposition

k-clique Enumeration

- How to efficiently enumerate k-cliques?
- Extend the graph orientation-based triangle enumeration algorithm
  - Orient the input undirected graph to be a directed graph
  - For each vertex *u* in *G*
    - Enumerate *(k-1)*-cliques in the subgraph of *G* induced by *u*'s out-neighbors

Each *k*-clique of *G* will be enumerated exactly once!

k-clique Enumeration



Algorithm 23: KClique-Oriented: enumerate all k-cliques in a graph [24]

```
Input: An undirected graph $G = (V, E)$, and an integer $k$
Output: All $k$-cliques in $G$

Compute the degeneracy ordering of $V$;
Construct the oriented graph $G^+ = (V, E^+)$ of $G$ with respect to the degeneracy ordering;
$C \leftarrow \emptyset$;
KClique-EnumO$(G^+, k, C)$;

Procedure KClique-EnumO$(G_k^+, k, C)$
if $k = 2$ then
    for each edge $(u, v) \in E(G_k^+)$ do
        Output clique $\{u, v\} \cup C$;
else
    for each vertex $u \in V(G_k^+)$ do
        $G_{k-1}^+ \leftarrow$ the subgraph of $G_k^+$ induced by $N_{G_k^+}^+(u)$;
        KClique-EnumO$(G_{k-1}^+, k-1, C \cup \{u\})$;
```

- All k-cliques can be enumerated in $O(k \times (\alpha(G))^{k-2} \times m)$ total time

Higher-order Densest Subgraph

- *k*-clique densest subgraph: find the subgraph g of G, such that the average number of k-cliques per vertex in g is the largest among all subgraphs of G

- The peeling algorithm can be extended to find a k-approximate k-clique densest subgraph

- The Goldberg's algorithm can be extended to find the k-clique densest subgraph exactly

Outline

- Background
- Core Decomposition
- Densest Subgraph Computation
- Higher-order Dense Subgraph Computation
- Future Directions

Future Directions

- How to do truss decomposition without hash tables?

- What is the relationship between dense subgraphs with different density (average degree) values?

- How to scale up nucleus decomposition and *k*-clique densest subgraphs for large *k* values?

- How to effectively and efficiently incorporate other information (such as attributes, temporal) into dense subgraph computation?

References

1. [Angel et al. 2012] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava. Dense subgraph main- tenance under streaming edge weight updates for real-time story identification. *PVLDB*, *5*(6):*574–585*, 2012.
2. [Batagelj and Zaversnik 2003] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. CoRR, cs.DS/0310049, 2003.
3. [Benson et al. 2016] A. Benson, D. Gleich, andJ. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.
4. [Boldi and Vigna 2004] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In Proc. of the Thirteenth International World Wide Web Conference (WWW 2004), pages 595–601, Manhattan, USA, 2004. ACM Press.
5. [Bonchi et al 2014] F. Bonchi, F. Gullo, A. Kaltenbrunner, Y. Volkovich: Core decomposition of uncertain graphs. KDD'14: 1316-1325
6. [Charikar 2000] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proc. APPROX'00*, pages 84–95, 2000.
7. [Chiba and Nishizeki 1985] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.
8. [Cohen 2008] J. Cohen. Trusses: Cohesive subgraphs for social network analysis, 2008.

References (cont')

9. [Danish et al 2018] M. Danisch, O. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *Proc. of WWW'18*, 2018.
10. [Danish et al 2017] M. Danisch, T.-H. H. Chan, M. Sozio: Large Scale Density-friendly Graph Decomposition via Convex Programming. WWW'17: 233-242
11. [Epasto et al 2015] A. Epasto, S. Lattanzi, M. Sozio: Efficient Densest Subgraph Computation in Evolving Graphs. WWW'15: 300-310
12. [Gallo et al 1989] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. SIAM J. Comput., 18(1):30–55, 1989.
13. [Giatsidis et al 2011] C. Giatsidis, D. M. Thilikos, M. Vazirgiannis: D-cores: Measuring Collaboration of Directed Graphs Based on Degeneracy. ICDM'11: 201-210
14. [Gionis and Tsourakakis 2015] A. Gionis, C. E. Tsourakakis: Dense Subgraph Discovery: KDD 2015 tutorial. KDD'15: 2313-2314
15. [Goldberg 1984] A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984
16. [Leskovec and Krevl 2014] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http: //snap.stanford.edu/data, June 2014.

References (cont')

17. [Malliaros et al 2016] F. D. Malliaros, A. N. Papadopoulos, M. Vazirgiannis: Core Decomposition in Graphs: Concepts, Algorithms and Applications. EDBT'16: 720-721
18. [Ortmann and Brandes 2014] M. Ortmann and U. Brandes. Triangle listing algorithms: Back from the diversion. In Proc. of ALENEX'14, pages 1–8, 2014.
19. [Qin et al 2015] L. Qin, R. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In Proc. of KDD'15, pages 965–974, 2015.
20. [Rossi and Ahmed 2015] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 2015.
21. [Sariyuce and Pinar 2016] A. E. Sariyu ̈ce and A. Pinar. Fast hierarchy construction for dense subgraphs. PVLDB, 10(3):97–108, 2016.
22. [Tsourakakis 2015] C. E. Tsourakakis. The k-clique densest subgraph problem. In Proc. of WWW'15, pages 1122–1132, 2015.
23. [Zhang et al 2017] Y. Zhang, J. X. Yu, Y. Zhang, L. Qin: A Fast Order-Based Approach for Core Maintenance. ICDE'17: 337-348

Springer Series in the Data Sciences

Lijun Chang · Lu Qin

Cohesive Subgraph Computation over Large Sparse Graphs

Algorithms, Data Structures, and Programming Techniques

This book is considered the first extended survey on algorithms and techniques for efficient cohesive subgraph computation. With rapid development of information technology, huge volumes of graph data are accumulated. An availability of rich graph data not only brings great opportunities for realizing big values of data to serve key applications, but also brings great challenges in computation. Using a consistent terminology, the book gives an excellent introduction to the models and algorithms for the problem of cohesive subgraph computation. The materials of this book are well organized from introductory content to more advanced topics while also providing well-designed source codes for most algorithms described in the book.

This is a timely book for researchers who are interested in this topic and efficient data structure design for large sparse graph processing. It is also a guideline book for new researchers to get to know the area of cohesive subgraph computation.

Mathematics

ISBN 978-3-030-03598-3

9 783030 035983

▸ springer.com

SSDS

Chang · Qin

Cohesive Subgraph Computation over Large Sparse Graphs

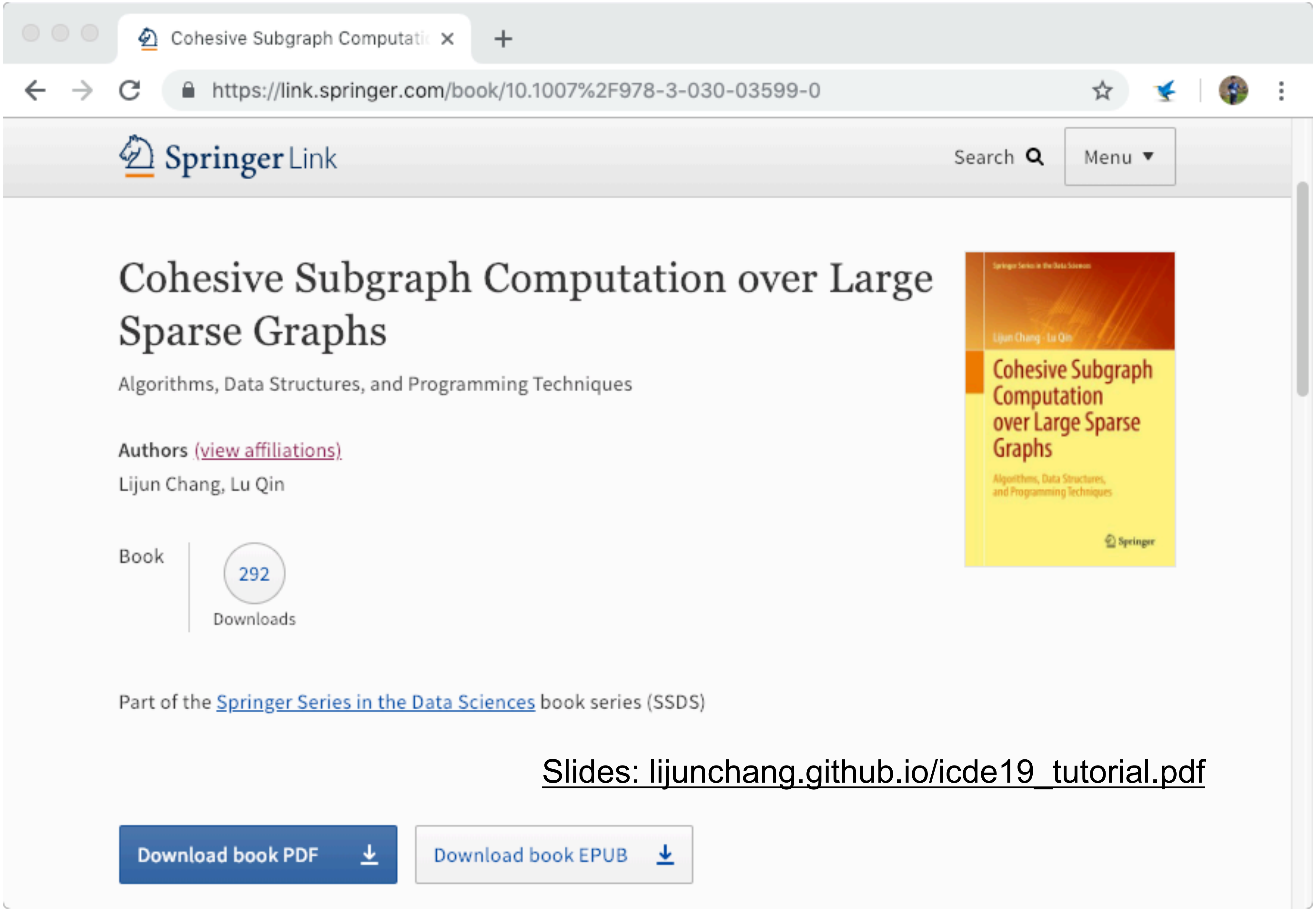
Springer Series in the Data Sciences

Lijun Chang · Lu Qin

Cohesive Subgraph Computation over Large Sparse Graphs

Algorithms, Data Structures, and Programming Techniques

Springer



Cohesive Subgraph Computati
https://link.springer.com/book/10.1007%2F978-3-030-03599-0
SpringerLink
Search
Menu
Springer Series in the Data Sciences
Lijun Chang · Lu Qin
Cohesive Subgraph Computation over Large Sparse Graphs
Algorithms, Data Structures, and Programming Techniques
Springer
Download book PDF
Download book EPUB

Cohesive Subgraph Computation over Large Sparse Graphs

Algorithms, Data Structures, and Programming Techniques

Authors (view affiliations)

Lijun Chang, Lu Qin

Book

292

Downloads

Part of the Springer Series in the Data Sciences book series (SSDS)

Slides: lijunchang.github.io/icde19_tutorial.pdf

PhD Recruitment at The University of Sydney

Lijun.Chang@sydney.edu.au