Cohesive Subgraph Computation over Large Graphs

Lijun Chang
DECRA Fellow @ Database Group
CSE, UNSW

Lijun
Outline

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
Graphs

- Graphs are everywhere
Big Graphs

Graphs are Big (Volume)

- 1.4 billion users in 2014
- 0.4 trillion relationships in 2014
- 302 million monthly active users
- 208 followers on average
- 2.1 billion webpages in 2000
- 15 billion edges in 2000
- 20 PB data/day in 2008
Properties of Real-world Graphs

- Real graphs are not *random graphs* (e.g., the Erdos-Renyi random graph model)
- have fascinating patterns and properties.
  - The *degree distribution* is skewed, following a power-law
  - The *average distance* is short (the small-world phenomenon)
  - *Edge density* is inhomogeneous - groups of vertices with high concentration of edges within them and low concentration between different groups
  - Globally sparse but locally dense
Cohesive Subgraph Computation

- Aims to identify the modules and, possibly, their hierarchical organization, by only using the information encoded in the graph topology.
- First attempt dates back to 1955 by Weiss and Jacobson searching for work groups within a government agency.
- Applications in various fields
  - Any context that information is encoded as a graph
Application domains

- 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*
  
- Related to functional modules such as cycles and pathways in *metabolic networks*
- Identify compartments in *food webs*
Cohesiveness Measures

- Minimum degree: **core decomposition**
- Minimum number of triangles each participates in: **truss decomposition**
- Edge connectivity: **edge connectivity-based decomposition**
- Clique: **maximal clique enumeration**
- Structural graph clustering
- …
Outline

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
Core Decomposition

- **k-core**
  - The largest subgraph in which every vertex has degree at least k within the subgraph

**Example:**
- Core number $c_1 = 1$
- Core number $c_1 = 2$
- Core number $c_1 = 3$

**Graph Degeneracy** $\delta^*(G) = 3$

- $G_0 = G$
- $G_1 = 1$-core of $G$
- $G_2 = 2$-core of $G$
- $G_3 = 3$-core of $G$

**Important property:**
- Fast and easy to compute
- Linear to the size of the graph
- Scalable to large scale graphs

**Note:**
The degeneracy and the size of the k-core provide a good indication of the cohesiveness of the graph

*Picture taken from Malliaros et al. 2016*
Core Decomposition

- Another Example

*Picture taken from Malliaros et al. 2016*
Peeling Algorithm

- **Basic idea**
  - Iteratively remove the weakest vertex (i.e., the one with the smallest degree)

0-core
- remove v6

1-core
- remove v1

2-core (also the max-core)
- If we remove another vertex, the graph collapses.

*Picture taken from Malliaros et al. 2016*
Time complexity

- Linear to the number of edges (i.e., $O(m)$)
- use a bin-sort like data structure to dynamically maintain the vertex with the smallest degree
Linear Time Algorithm

Picture taken from Malliaros et al. 2016
Linear Time Algorithm

The removal of node 9 leads to decreasing the degree of node 7.
Linear Time Algorithm

The removal of node 9 leads to decreasing the degree of node 7

Node 7 must change bin
Linear Time Algorithm

The removal of node 9 leads to decreasing the degree of node 7.

Picture taken from Malliaros et al. 2016
Linear Time Algorithm

The removal of node 9 leads to decreasing the degree of node 7
For Speeding Up Algorithms

- Densest subgraph detection
  - 2-approximation of the densest subgraph. Density is the average degree
  - prune vertices that are not in the densest subgraph, thus speed up the algorithm

- K-edge connected component computation.
  - A k-edge connected component is a subgraph of a k-core
Influential Spreaders

Most efficient spreaders are located within the $k$-core of the network

Outline

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
Truss Decomposition

- K-core decomposition often returns a relatively large number of candidate influential spreaders.
  - Only a small fraction corresponds to truly highly influential vertices.
- k-truss decomposition further refines the set of the most influential vertices.
Truss Decomposition

- **K-truss**
  - The maximal subgraph in which every edge participates in \( k-2 \) triangles

*Picture taken from Rossi et al. 2015*
**K-truss vs k-core**

- k-truss is a subgraph of $(k-1)$-core
- k-truss represents the nucleus of a k-core filtering out less important information

*Picture taken from Rossi et al. 2015*
Truss Decomposition Algorithm

- Extend the peeling algorithm, which is designed for core decomposition, to truss decomposition
  - That is, virtually transform the graph $G$ to a new graph $G'$
  - Each edge $(u,v)$ in $G$ corresponds to a vertex in $G'$
  - Two vertices in $G'$ are connected by an edge if 1) their corresponding edges in $G$ have one common vertex, and 2) the three vertices of the two corresponding edges form a triangle in $G$

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

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
A graph is $k$-edge connected if it is still connected after removing any set of $(k-1)$ edges from it.

2-edge connected
k-edge Connected Components

- Given a graph $G$ and an integer $k$, computing all maximal subgraphs of $G$ that are $k$-edge connected.
Observation

- A graph is not k-edge connected, iff \( \exists \) a set C of edges \(|C| < k\) whose removal disconnects the graph.

Example: the above graph is not 3-edge connected: cut \{(9,11), (5,12)\}
Algorithm

- Given a graph $G$ and an integer $k$
  - If $G$ is not $k$-edge connected, find a cut $C$ with cardinality $< k$
    - Remove edges in $C$ from $G$, then the result will be two connected subgraphs
    - Find $k$-edge connected components for each connected subgraph
  - Otherwise, report that this subgraph is a $k$-edge connected component
Example (k=3)
Optimizations

- Optimizations [Chang et al. SIGMOD’13]
  - k-core pruning
  - Find multiple cuts at the same time
  - Fast compute a cut

- The time complexity of computing k-edge connected components is $O(h*l*|E|)$
  - $h$ and $l$ are bounded small constants
**Edge-connectivity based decomposition**

[Chang et al. SIGMOD’15]

- **Compute** $sc(u, v)$ for all edges $(u, v)$ in the graph.
  - $sc(u, v)$: the maximum $k$ such that a $k$-edge connected component containing edge $(u, v)$

- Use the existing techniques for computing $k$-edge connected component with two optimizations.
  - **Batch processing:** assign $sc(u, v)$ values for all edges in $k$-edge connected components
  - **Computation sharing:** all edges removed during computing $k$-edge connected components are also removed when computing $(k+1)$-edge connected components

**Time complexity:** $O(\alpha(G) \times h \times l \times |E|)$, where $\alpha(G)$ is the arboricity of a graph $G$. 

---
Outline

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
Maximal Clique Enumeration

Algorithmica’13: Fast Maximal Clique Enumeration in Sparse Graphs
Backtracking Algorithm

- Compute the Maximum Clique is NP-hard
- Backtracking
  - starting with a vertex $C = \{u\}$ and all its neighbors as the candidate $P$
  - Recursively add a vertex from $P$ to $C$, and reduce $P$ to the subset that are adjacent to all vertices of $C$
  - A maximal clique is found if $P$ is empty
- A clique is contained in the neighborhood-subgraph of a vertex
  - In practice, the maximum clique can also be found efficiently for real graphs
Outline

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
Structural Graph Clustering

- Structural graph clustering: SCAN [Xu+, KDD’07]
  - Identifies clusters, hubs, and outliers at the same time
  - Mimics DBSCAN [Ester+, KDD’96] for clustering spatial data

![Example structural graph clustering](image)
A Cluster = Cores + Borders

Core: vertices that are structure-similar to many other vertices
Border: vertices that are not core but are structure-similar to a core

Structural Similarity: \[ \sigma(u, v) = \frac{|N[u] \cap N[v]|}{\sqrt{d[u] \cdot d[v]}}. \]

- Two vertices \( u \) and \( v \) are structure-similar if
  - Connected
  - Structural similarity \( \geq \varepsilon \) (a given similarity threshold)
  - Many: \( \geq \mu \) (a given size threshold)
Example ($\varepsilon=0.0001$, $\mu=3$)
pSCAN (Chang et al. ICDE’17)

- A two-step framework + optimization techniques
  - **Step-I:** Cluster core vertices
    - Transitivity: if core $u$ and core $v$ are in the same cluster, core $v$ and core $w$ are in the same cluster, then core $u$ and core $w$ are in the same cluster
    - Reduces the number of structural similarity computations
  - **Step-II:** Cluster non-core vertices
    - A non-core vertex belongs to the same cluster of a set of core vertices if it is structure-similar to one of the core vertices
- Time complexity is $O(\alpha(G) \times m)$
- pSCAN is worst-case optimal
Wrap up

- Introduction
- Core Decomposition
- Truss Decomposition
- Edge-connectivity based Decomposition
- Maximal Clique Enumeration
- Graph Structural Clustering
Thank you!

Questions?

ljchang@cse.unsw.edu.au