Mining Large Dynamic Graphs and Tensors

Kijung Shin
Ph.D. Candidate
Carnegie Mellon University
(kijungs@cs.cmu.edu)
Thesis Committee

• Prof. Christos Faloutsos (Chair)
• Prof. Tom M. Mitchell
• Prof. Leman Akoglu
• Prof. Philip S. Yu
What Do Real Graphs Look Like?

• Part 1 (Chapters 3 - 8)
How to Spot Anomalies?

- Part 2 (Chapters 9 - 13)
How to Model Behavior?

• Part 3 (Chapters 14 - 15)
Graphs are Everywhere!
Graphs are Large and Dynamic

- **Large**: many nodes, more edges
  - 40B+ web pages
  - 500M+ products

- **Dynamic**: additions/deletions of nodes and edges
  - 2B+ active users
  - 5M+ articles
.. and with Rich Side Information

- **Rich**: timestamps, scores, text, etc.
Simple Graphs are Matrices

Graph

Adjacency Matrix

\[
\begin{array}{ccc}
0 & 0 & 0 \\
1 & 1 & 1 \\
1 & 0 & 0 \\
1 & 1 & 0 \\
\end{array}
\]
Rich Graphs are Tensors

- **Tensors**: multi-dimensional array

- **3-order tensor**
  - (3-dimensional array)

- **4-order tensor**
  - Stars 🌟🌟🌟 (4-order tensor)

- **5-order tensor**
  - Satisfied 😊 (5-order tensor)

3-order tensor
(3-dimensional array)
Thesis Goal and Focus

• Goal:

To Fully Understand and Utilize Large Dynamic Graphs and Tensors on User Behavior

• Our Focus: To Develop Scalable Algorithms for
  ▪ T1. Structure Analysis (Part 1)
  ▪ T2. Anomaly Detection (Part 2)
  ▪ T3. Behavior Modeling (Part 3)
Tasks and Their Relation

- Given large dynamic graphs or tensors,

- "How do they look like?"

- "How to model behavior?"

- "How to spot anomalies?"
Our Tools for Scalability

• We design (sub) linear algorithms

Approx. Sampling Streaming Out-of-core Parallel

• Running on big data platforms

• Exploiting empirical patterns in data
  ◦ locality, power-laws, etc.
# Organization of the Thesis

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs</strong></td>
<td><strong>Tensors</strong></td>
<td></td>
</tr>
<tr>
<td>Triangle Count ($§§$ 3-6)</td>
<td>Summarization ($§$ 7)</td>
<td>Summarization ($§$ 8)</td>
</tr>
<tr>
<td>Anomalous Subgraph ($§$ 9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purchase Behavior ($§$ 14)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Focuses of This Presentation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Count (§§ 3-6)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summarization (§ 7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tensors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summarization (§ 8)</td>
<td>Dense Subtensors (§§ 10-13)</td>
<td>Progression (§ 15)</td>
</tr>
</tbody>
</table>
Roadmap

• **T1. Structure Analysis (Part 1)**
• **T2. Anomaly Detection (Part 2)**
• **T3. Behavior Modeling (Part 3)**
  • Future Directions
  • Conclusion
“Given a large graph (or tensor), how can we analyze its structure?

Input graph

Basic statistics

Structure measures

- density
- clustering coefficients
- transitivity ratio
- triangle connectivity

T1-1. Triangle Counting

Summary graph

T1-2. Summarization
T1-1. Triangle Counting (§§ 4-6)

“Given a large dynamic graph, how can we track the count of triangles accurately with sub-linear memory?”
How can we exploit **temporal patterns?** (§4)

How can we make good use of **multiple machines?** (§5)

How can we handle **removed edges?** (§6)
Roadmap

- T1. Structure Analysis (Part 1)
  - T1.1 Triangle Counting
    - Handling Deletions (§6) <<
    - ...
    - ...

- T2. Anomaly Detection (Part 2)

- T3. Behavior Modeling (Part 3)
  - Future Directions
  - Conclusions

K. Shin, J. Kim B. Hooi, C. Faloutsos, “Think before You Discard: Accurate Triangle Counting in Graph Streams with Deletions”, ECML/PKDD 2018
Triangles in a Graph

- A triangle is 3 nodes connected to each other
- The count of triangles is an important primitive
  - Applications:
    - community detection, spam detection, link prediction
  - Structure measures:
    - transitivity ratio, clustering coefficients, trussness
Remaining Challenge

• Counting triangles in real-world graphs
  ◦ Large: not fitting in main memory
  ◦ Fully dynamic: both growing and shrinking

Online social networks: Facebook, Twitter
Web
Citation networks
Call networks
Previous Work

- **Given**: a large and fully-dynamic graph
- **To estimate**: the count of triangles accurately

<table>
<thead>
<tr>
<th></th>
<th>Large Graph</th>
<th>Fully dynamic Graph</th>
<th>Accurate</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASCOT [LJK18]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Triest-IMPR [DERU17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>WRS [Shi17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ESD [HS17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Triest-FD [DERU17]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>ThinkD (Proposed)</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Our Contribution: ThinkD

- We develop ThinkD (Think before You Discard):

  - Fast & Accurate: outperforming competitors
  - Scalable: linear data scalability
  - Theoretically Sound: unbiased estimates
Roadmap

• T1. Structure Analysis (Part 1)
  ◦ T1.1 Triangle Counting
    ▪ Handling Deletions (§6)
      ◦ Problem Definition <<
      ◦ Proposed Method: ThinkD
      ◦ Experiments
    ▪ ...

• T2. Anomaly Detection (Part 2)

• T3. Behavior Modeling (Part 3)
  ◦ ...

Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
Fully Dynamic Graph Stream

- Our model for a large and fully-dynamic graph
- Discrete time $t$, starting from 1 and ever increasing
- At each time $t$, a change in the input graph arrives
  - change: either an insertion or deletion of an edge

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change (given)</td>
<td>$+(a, b)$</td>
</tr>
<tr>
<td>Graph (uninitialized)</td>
<td>$a \rightarrow b$</td>
</tr>
</tbody>
</table>
**Fully Dynamic Graph Stream**

- Our model for a large and fully-dynamic graph
- Discrete time $t$, starting from 1 and ever increasing
- At each time $t$, a *change* in the input graph arrives
  - *change*: either an *insertion* or *deletion* of an edge

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change (given)</td>
<td>$(a, b)$</td>
<td>$(a, c)$</td>
</tr>
<tr>
<td>Graph (unmaterialized)</td>
<td>$a - b$</td>
<td>$a - b - c$</td>
</tr>
</tbody>
</table>
Fully Dynamic Graph Stream

- Our model for a large and fully-dynamic graph
- Discrete time $t$, starting from 1 and ever increasing
- At each time $t$, a change in the input graph arrives
  - change: either an insertion or deletion of an edge

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change (given)</td>
<td>$(a, b)$</td>
<td>$(a, c)$</td>
<td>$(b, c)$</td>
</tr>
<tr>
<td>Graph (unmarried)</td>
<td>$ab$</td>
<td>$ac$</td>
<td>$bc$</td>
</tr>
</tbody>
</table>

Fully Dynamic Graph Stream

• Our model for a large and fully-dynamic graph
• Discrete time $t$, starting from 1 and ever increasing
• At each time $t$, a change in the input graph arrives
  • change: either an insertion or deletion of an edge

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change (given)</td>
<td>$(a, b)$</td>
<td>$(a, c)$</td>
<td>$(b, c)$</td>
<td>$-(a, b)$</td>
</tr>
<tr>
<td>Graph (unmarried)</td>
<td>$a \quad b$</td>
<td>$a \quad b \quad c$</td>
<td>$a \quad b \quad c$</td>
<td>$a \quad b \quad c$</td>
</tr>
</tbody>
</table>

Fully Dynamic Graph Stream

• Our model for a large and fully-dynamic graph
• Discrete time $t$, starting from 1 and ever increasing
• At each time $t$, a change in the input graph arrives
  ◦ change: either an insertion or deletion of an edge

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change (given)</td>
<td>$(a, b)$</td>
<td>$(a, c)$</td>
<td>$(b, c)$</td>
<td>$-(a, b)$</td>
<td>$(b, d)$</td>
<td>...</td>
</tr>
<tr>
<td>Graph (unmaterialized)</td>
<td>$a$ $b$</td>
<td>$a$ $c$ $b$</td>
<td>$a$ $b$ $c$</td>
<td>$a$ $b$ $c$ $d$</td>
<td>$a$ $b$ $c$ $d$</td>
<td>...</td>
</tr>
</tbody>
</table>
## Fully Dynamic Graph Stream

- Our model for a large and fully-dynamic graph
- Discrete time $t$, starting from 1 and ever increasing
- At each time $t$, a \textit{change} in the input graph arrives
  - \textit{change}: either an \textit{insertion} or \textit{deletion} of an edge

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change (given)</td>
<td>+(a, b)</td>
<td>+(a, c)</td>
<td>+(b, c)</td>
<td>−(a, b)</td>
<td>+(b, d)</td>
<td>...</td>
</tr>
<tr>
<td>Graph (unmaterialized)</td>
<td><img src="image" alt="Graph Diagram" /></td>
<td><img src="image" alt="Graph Diagram" /></td>
<td><img src="image" alt="Graph Diagram" /></td>
<td><img src="image" alt="Graph Diagram" /></td>
<td><img src="image" alt="Graph Diagram" /></td>
<td>...</td>
</tr>
</tbody>
</table>

Not Materialized
Problem Definition

• **Given:**
  ◦ a fully-dynamic graph stream *(possibly infinite)*
  ◦ memory space *(finite)*

• **Estimate:** the count of triangles

• **To Minimize:** estimation error

<table>
<thead>
<tr>
<th>Time $t$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changes</td>
<td>$(a, b)$</td>
<td>$(a, c)$</td>
<td>$(b, c)$</td>
<td>$-(a, b)$</td>
<td>$(b, d)$</td>
<td>…</td>
</tr>
<tr>
<td># Triangles</td>
<td>🎉</td>
<td>🎉</td>
<td>🎉</td>
<td>🎉</td>
<td>🎉</td>
<td>…</td>
</tr>
</tbody>
</table>

Given (input)  
Estimate (output)
**Roadmap**

- **T1. Structure Analysis (Part 1)**
  - **T1.1 Triangle Counting**
    - **Handling Deletions (§6)**
      - Problem Definition
      - **Proposed Method: ThinkD <<**
      - Experiments
    - ...

- **T2. Anomaly Detection (Part 2)**

- **T3. Behavior Modeling (Part 3)**
  - Future Directions
  - Conclusions
Overview of ThinkD

• Maintains and updates $\Delta$
  ◦ Number of (non-deleted) triangles that it has observed

• How it processes an insertion:

- **arrive**: an insertion of an edge arrives
- **count**: count new triangles and increase $\Delta$
- **test**: toss a coin
- **store**: store the edge in memory
Overview of ThinkD (cont.)

- Maintains and updates $\hat{\Delta}$
  - Number of (non-deleted) triangles that it has observed
- How it processes an deletion:

**Flowchart:**

- **arrive**: a deletion of an edge arrives
- **count**: count deleted triangles and decrease $\hat{\Delta}$
- **test**: test whether the edge is stored in memory
- **delete**: delete the edge in memory
Why is ThinkD Accurate?

- **ThinkD** (Think before You Discard):
  - *every* arrived change is used to update $\Delta$

- **Triest-FD** [DERU17]:
  - *some* changes are discarded without being used to update $\Delta$

\[\text{arrive} \rightarrow \text{count: update } \Delta \rightarrow \text{test} \rightarrow \text{store / delete} \]

\[\text{arrive} \rightarrow \text{test} \rightarrow \text{count: update } \Delta \rightarrow \text{store / delete} \]

No (discard) $\rightarrow$ information loss!
Two Versions of ThinkD

Q1: How to test in the test step
Q2: How to estimate the count of all triangles from $\hat{\Delta}$

- **ThinkD-FAST**: simple and fast
  - independent Bernoulli trials with probability $p$

- **ThinkD-ACC**: accurate and parameter-free
  - random pairing [GLH08]
Unbiasedness of ThinkD-FAST

- $\frac{\hat{\Delta}}{p^2}$: estimated count of \textit{all triangles}
- $\Delta$: true count of \textit{all triangles}

[Theorem 1] At any time $t$,

$$\mathbb{E} \left[ \frac{\hat{\Delta}}{p^2} \right] = \Delta$$

Unbiased estimate of $\Delta$

- Proof and a variance of $\frac{\hat{\Delta}}{p^2}$: see the thesis
ThinkD-ACC: More Accurate

• Disadvantage of ThinkD-FAST:
  ◦ setting the parameter $p$ is not trivial
    ▪ small $p \rightarrow$ underutilize memory
      $\rightarrow$ inaccurate estimation
    ▪ large $p \rightarrow$ out-of-memory error

• ThinkD-ACC uses Random Pairing [RLH08]
  ◦ always utilizes memory as fully as possible
  ◦ gives more accurate estimation
Scalability of ThinkD

• Let $k$ be the size of memory
• For processing $t$ changes in the input stream,

[ Theorem 2 ] The time complexity of ThinkD-ACC is \( O(k \cdot t) \) linear in data size

[ Theorem 3 ] If $p = O\left(\frac{k}{t}\right)$, the time complexity ThinkD-FAST is \( O(k \cdot t) \)
Advantages of ThinkD

- **Fast & Accurate:** outperforming competitors
- **Scalable:** linear data scalability (Theorems 2 & 3)
- **Theoretically Sound:** unbiased estimates (Theorem 1)
Roadmap

• T1. Structure Analysis (Part 1)
  ◦ T1.1 Triangle Counting
    ▪ Handling Deletions (§6)
      ◦ Problem Definition
      ◦ Proposed Method: ThinkD
      ◦ Experiments <<

  ▪ ...

• T2. Anomaly Detection (Part 2)

• T3. Behavior Modeling (Part 3)
  ◦ ...

Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
Experimental Settings

- **Competitors:** *Triest-FD* [DERU17] & *ESD* [HS17]
  - triangle counting in fully-dynamic graph streams
- **Implementations:**
  - Java 8
- **Datasets:**
  - insertions (edges in graphs) + deletions (random 20%)
  - Synthetic (100B edges)
  - Social Networks (1.8B+ edges, ...)
  - Citation (16M+)
  - Web (6M+)
  - Trust (0.7M+)
EXP1. Variance Analysis

ThinkD is accurate with small variance

Number of Processed Changes

- dataset: Facebook

EXP2. Scalability [THM 2 & 3]

ThinkD is scalable

- dataset: 100 billion changes
**EXP3. Space & Accuracy**

*ThinkD outperforms its best competitors*

- **Dataset**:
  - Graph
  - Tensor

- **Part**:
  - Part 1
  - Part 2
  - Part 3

- **Ratio**:
  - Memory budget
  - Estimation error

- **Comparisons**:
  - Triest-FD
  - ThinkD-FAST
  - ESD

- **Accuracy**:
  - 1.9 - 4x more accurate
EXP4. Speed & Accuracy

ThinkD outperforms its best competitors

- dataset: 😊
Advantages of ThinkD

- **Fast & Accurate:** outperforming competitors
- **Scalable:** linear data scalability
- **Theoretically Sound:** unbiased estimates
Summary of §6

• We propose **ThinkD (Think Before you Discard)**
  ◦ for accurate *triangle counting*
  ◦ in *large* and *fully-dynamic* graphs

✔ **Fast & Accurate:** outperforming competitors

✔ **Scalable:** linear data scalability

✔ **Theoretically Sound:** unbiased estimates
# Organization of the Thesis (Recall)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Count (§§ 3-6)</td>
<td>Anomalous Subgraph (§ 9)</td>
<td>Purchase Behavior (§ 14)</td>
</tr>
<tr>
<td>Summarization (§ 7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tensors</strong></td>
<td>Summarization (§ 8)</td>
<td>Dense Subtensors (§§ 10-13)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Progression (§ 15)</td>
</tr>
</tbody>
</table>

- **Graphs**
  - Triangle Count (§§ 3-6)
  - Summarization (§ 7)

- **Tensors**
  - Summarization (§ 8)
  - Dense Subtensors (§§ 10-13)

- **Part 1. Structure Analysis**

- **Part 2. Anomaly Detection**

- **Part 3. Behavior Modeling**
“Given a web-scale graph or tensor, how can we succinctly represent it?”

- §7: Summarizing Graphs
- §8: Summarizing Tensors (via Tucker Decomposition)
  - External-memory algorithm with $1,000 \times$ improved scalability
Roadmap

- T1. Structure Analysis (Part 1)
  - ...
  - T1-2. Summarization (§§ 7-8)
    - Summarizing Graphs (§ 7)
      - Problem Definition <<
      - Proposed Method: SWeG
      - Experiments
  - ...

- T2. Anomaly Detection (Part 2)
- T3. Behavior Modeling (Part 3)
  - ...

K. Shin, A. Ghoting, M. Kim, and H. Raghavan, “SWeG: Lossless and Lossy Summarization of Web-Scale Graphs”, WWW 2019
Graph Summarization: Example

Input Graph (w/ 9 edges)

Output (w/ 6 edges)
Graph Summarization [NRS08]

- **Given:** an input graph
- **Find:**
  - a summary graph
  - positive and negative residual graphs
- **To Minimize:** the edge count (≈ description length)
Restoration: Example

\[
(a, b), (c, d, e), (f, g) \quad \rightarrow \quad (a, b), (c, d, e), f, g
\]

Summarized Graph (w/ 6 edges)

\[
(a, b), (c, d, e), f, g \quad \rightarrow \quad (a, b), (c, d, e), f, g
\]

Restored Graph (w/ 9 edges)
Why Graph Summarization?

• Summarization:
  ◦ the summary graph is easy to visualize and interpret

• Compression:
  ◦ support efficient neighbor queries
  ◦ applicable to lossy compression
  ◦ combinable with other graph compression techniques
  ▪ the outputs are also graphs

Summary Graph

Residual Graph (Positive)
+ (d, g)

Residual Graph (Negative)
− (a, d)
− (c, e)

discussed in the thesis
Challenge: Scalability!

Compression Performance

Good

Bad

Maximum Size of Graphs

VoG [KKVF14]
Greedy [NSR08]
Randomized [NSR08]
SAGS [KNL15]
SWeG

10,000 ×
Our Contribution: SWeG

• We develop **SWeG** (**Summarizing Web-scale Graphs**):

- Fast with Concise Outputs
- Memory Efficient
- Scalable
Roadmap

- T1. Structure Analysis (Part 1)
  - ...  
  - T1-2. Summarization (§§ 7-8)
    - Summarizing Graphs (§ 7)
      - Problem Definition
      - Proposed Method: SWeG
      - Experiments
  - ...

- T2. Anomaly Detection (Part 2)

- T3. Behavior Modeling (Part 3)
  - ...

Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
Terminologies

Summary Graph $S$

\[
\{a, b\} = A \\
\{c, d, e\} = B \\
\{f, g\} = C
\]

Residual Graph $R$

Positive Residual Graph $R^+$

\[
+ (d, g)
\]

Negative Residual Graph $R^-$

\[
- (a, d) \\
- (c, e)
\]

\[
\text{Saving}(A, B) := 1 - \frac{\text{Cost}(A \cup B)}{\text{Cost}(A) + \text{Cost}(B)}
\]

Encoding cost of $A$

Encoding cost of $B$

Encoding cost when $A$ and $B$ are merged
Overview of SWeG

• **Inputs:**
  - input graph $G$
  - number of iterations $T$

• **Outputs:**
  - summary graph $S$
  - residual graph $R$ (or $R^+$ and $R^-$)

• **Procedure:**

- **S0:** Initializing Step
- repeat $T$ times
  - **S1-1:** Dividing Step
  - **S1-2:** Merging Step
- **S2:** Compressing Step (optional)
Overview: **Initializing Step**

Summary Graph $\mathbf{S} = \mathbf{G}$  
Residual Graph $\mathbf{R} = \emptyset$

\[
\begin{align*}
A &= \{a\} & D &= \{d\} & F &= \{f\} \\
B &= \{b\} & G &= \{g\} & E &= \{e\}
\end{align*}
\]

- **S0: Initializing Step <<**
- repeat $T$ times
  - S1-1: Dividing Step
  - S1-2: Merging Step
- S2: Compressing Step (optional)
Overview: **Dividing Step**

- Divides super nodes into groups
  - **MinHashing (used)**, EigenSpoke, Min-Cut, etc.

  \[
  C = \{c\} \\
  A = \{a\} \\
  B = \{b\} \\
  D = \{d\} \\
  E = \{e\} \\
  F = \{f\} \\
  G = \{g\}
  \]

- S0: Initializing Step
- repeat $T$ times
  - **S1-1: Dividing Step**
  - S1-2: Merging Step
  - S2: Compressing Step (optional)
Overview: Merging Step

• Merge some supernodes within each group if \( Saving > \theta(t) \)

\[
C = \{c\} \quad D = \{d, e\} \quad F = \{f, g\}
\]

\[
A = \{a, b\}
\]

• S0: Initializing Step
• repeat \( T \) times
  • S1-1: Dividing Step
  • S1-2: Merging Step <<
• S2: Compressing Step (optional)
Overview: Merging Step (cont.)

Summary Graph $S$

$A = \{a, b\}$  \hspace{2cm} $C = \{c\}$  \hspace{2cm} $D = \{d, e\}$  \hspace{2cm} $F = \{f, g\}$

Residual Graph $R$

$+ (d, g)$  \hspace{2cm} $- (a, d)$

- S0: Initializing Step
- repeat $T$ times
  - S1-1: Dividing Step
  - S1-2: Merging Step
- S2: Compressing Step (optional)
Overview: **Dividing Step**

- Divides super nodes into groups

\[
\begin{align*}
F &= \{f, g\} \\
A &= \{a, b\} \\
C &= \{c\} \\
D &= \{d, e\}
\end{align*}
\]

- **S0:** Initializing Step
- repeat $T$ times
  - **S1-1:** Dividing Step <<
  - **S1-2:** Merging Step
  - **S2:** Compressing Step (optional)
Overview: **Merging Step**

- Merge some supernodes within each group if \( Saving > \theta^{(t)} \)

\[ \begin{align*}
F &= \{f, g\} \\
A &= \{a, b\} \\
C &= \{c, d, e\}
\end{align*} \]

- **S0:** Initializing Step
- repeat \( T \) times
  - **S1-1:** Dividing Step
  - **S1-2:** Merging Step \(<< \)
- **S2:** Compressing Step (optional)
Overview: Merging Step (cont.)

Summary Graph $S$

- $A = \{a, b\}$
- $B = \{c, d, e\}$
- $F = \{f, g\}$

Residual Graph $R$

- $+ (d, g)$
- $- (a, d)$
- $- (c, e)$

- **S0**: Initializing Step
- **repeat** $T$ **times**
  - **S1-1**: Dividing Step
  - **S1-2**: Merging Step
- **S2**: Compressing Step (optional)
Overview: Merging Step (cont.)

- Merge some supernodes within each group if $Saving > \theta^{(t)}$
- Decreasing $\theta^{(t)} = (1 + t)^{-1}$
  - $exploration$ of other groups
  - $exploitation$ within each group
  - $\sim 30\%$ better compression than $\theta^{(t)} = 0$

- S0: Initializing Step
- repeat $T$ times
  - S1-1: Dividing Step
  - **S1-2: Merging Step **
- S2: Compressing Step (optional)
Overview: **Compressing Step**

- Compress each output graph ($S$, $R^+$ and $R^-$)
- Use any off-the-shelf graph-compression algorithm
  - Boldi-Vigna [BV04]
  - VNMiner [BC08]
  - Graph Bisection [DKKO+16]

- **S0: Initializing Step**
- repeat $T$ times
  - S1-1: Dividing Step
  - S1-2: Merging Step
- **S2: Compressing Step (optional)**
Parallel & Distributed Processing

• **Map** stage: compute *min hashes* in *parallel*
• **Shuffle** stage: divide super nodes using *min hashes*
• **Reduce** stage: process groups independently in *parallel*

No need to load the entire graph in memory!

\[
A = \{a\}, \quad B = \{b\}, \quad C = \{c\}
\]

\[
D = \{d\}, \quad E = \{e\}
\]

\[
F = \{f\}, \quad G = \{g\}
\]
Roadmap

• T1. Structure Analysis (Part 1)
  ◦ ...
  ◦ T1-2. Summarization (§§ 7-8)
    ▪ Summarizing Graphs (§ 7)
      ◦ Problem Definition
      ◦ Proposed Method: SWeG
      ◦ Experiments <<
  ◦ ...

• T2. Anomaly Detection (Part 2)

• T3. Behavior Modeling (Part 3)
  ◦ ...

Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
Experimental Settings

- 13 real-world graphs (10K - 20B edges)

Graph summarization algorithms:
  - Greedy [NRS08], Randomized [NSR08], SAGS [KNL15]

Implementations: Java & Hadoop
EXP1. Speed and Compression

*SWeG outperforms its competitors*

- dataset: dblp

- Graph
- Tensor
- §4 / §5 / §6 / §7

Part 1 / Part 2 / Part 3

- 370 - 4,490 faster
Advantages of SWeG (Recall)

- ✔ Fast with Concise Outputs
- □ Memory Efficient
- □ Scalable
EXP2. Memory Efficiency

*SWeG loads \( \leq 0.1 - 4\% \) of edges in main memory at once

<table>
<thead>
<tr>
<th>Graph</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblp</td>
<td>108X</td>
</tr>
<tr>
<td></td>
<td>42X</td>
</tr>
<tr>
<td>WWW</td>
<td>56X</td>
</tr>
<tr>
<td>LIVEJOURNAL</td>
<td>1209X</td>
</tr>
<tr>
<td>PATENTED</td>
<td>294X</td>
</tr>
<tr>
<td>HOLLYWOOD</td>
<td>139X</td>
</tr>
<tr>
<td>WWW</td>
<td>27X</td>
</tr>
</tbody>
</table>

Advantages of SWeG (Recall)

- Fast with Concise Outputs
- Memory Efficient

☐ Scalable
EXP3. Effect of Iterations

About 20 iterations are enough

Graph / Tensor

EXP4. Data Scalability

SWeG is linear in the number of edges

\[ \geq 20 \text{ billion edges} \]
EXP5. Machine Scalability

SWeG scales up

**Graph/Part 1**

** Tensor/Part 2**

§/§/§/§
Advantages of SWeG (Recall)

- Fast with Concise Outputs
- Memory Efficient
- Scalable
Summary of §7

- We propose **SWeG (Summarizing Web Graphs)**
  - for summarizing large-scale graphs

- **Fast with Concise Outputs**
- **Memory Efficient**
- **Scalable**
Contributions and Impact (Part 1)

- **Triangle counting algorithms** [ICDM17, PKDD18, PAKDD18]
- **Summarization algorithms** [WSDM17, WWW19]
- **Patent** on *SWeG*: filed by LinkedIn Inc.
- **Open-source software**: downloaded *82 times*

---

**Graphs and Diagrams**

- Global Error vs. Ratio of Stored Edges
  - **ThinkD**: 2.5X
- Rel. Size of Summary vs. Elapsed Time (sec)
  - **SWeG**: 650X

---

**GitHub Repository**

github.com/kijungs/SWeG

---

**Patent Information**

ThinkD
SWeG

---

**Additional Notes**

- Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
### Organization of the Thesis (Recall)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs</strong></td>
<td><strong>Tensors</strong></td>
<td></td>
</tr>
<tr>
<td>Triangle Count (§§ 3-6)</td>
<td>Summarization (§ 7)</td>
<td>Summarization (§ 8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Anomalous Subgraph (§ 9)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Progression (§ 15)</td>
</tr>
<tr>
<td></td>
<td>Dense Subtensors (§§ 10-13)</td>
<td></td>
</tr>
</tbody>
</table>
“How can we detect anomalies or fraudsters in large dynamic graphs (or tensors)?”

**Hint:** fraudsters tend to form dense subgraphs.
T2-1. Utilizing Patterns

- T2-1. Patterns and Anomalies in Dense Subgraphs (§ 9)

“What are patterns in dense subgraphs?”
“What are anomalies deviating from the patterns?”

K. Shin, T. Eliassi-Rad, C. Faloutsos, “Patterns and Anomalies in k-Cores of Real-world Graphs with Applications”, KAIS 2018 (formerly, ICDM 2016)
T2-2. Utilizing Side Information

Accounts

(or/+ IPs)
(or/+ Ratings)
(Timestamps)

Items

Sparse

Part 1

Part 2

Part 3

Graph

Tensor

§11 / §12 / §13 / §14
“How can we detect dense subtensors in large dynamic data?”

• T2-2. Detecting Dense Subtensors (§§ 11-13)
  ◦ In-memory Algorithm (§ 11)
  ◦ Distribute Algorithm for Web-scale Tensors (§ 12)
  ◦ Incremental Algorithms for Dynamic Tensors (§ 13)
Contributions and Impact (Part 2)

Patterns in dense subgraphs [ICDM16]
- Award: best paper candidate at ICDM 2016
- Class: MIT Massachusetts Institute of Technology

Algorithms for dense subtensors [PKDD16, WSDM17, KDD17]
- Real-world usage: NAVER

Open-source software: downloaded 257 times

Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
### Organization of the Thesis (Recall)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Count</td>
<td></td>
<td>Purchase Behavior</td>
</tr>
<tr>
<td>(§§ 3-6)</td>
<td></td>
<td>(§ 14)</td>
</tr>
<tr>
<td>Summarization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(§ 7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tensors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summarization</td>
<td></td>
<td>Progression</td>
</tr>
<tr>
<td>(§ 8)</td>
<td></td>
<td>(§ 15)</td>
</tr>
<tr>
<td>Dense Subtensors</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(§§ 10-13)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
T3. Behavior Modeling (Part 3)

“How can we model the behavior of individuals in graph and tensor data?”

Social Network

Behavior Log on Social Media

• T3-1. Modeling Purchase Behavior in a Social Network (§14)

• T3-2. Modeling Progression of Users of Social Media (§15)

“How do users evolve over time on social media?”
Roadmap

• T1. Structure Analysis (Part 1)
• T2. Anomaly Detection (Part 2)
• T3. Behavior Modeling (Part 3)
  ◦ T3-1. Modeling Purchases (§14) <<
  ◦ ...

• Future Directions
• Conclusions

Sharable Goods: Question

“What do they have in common?”

Portable crib  IKEA toolkit  DVDs
Sharable Goods: Properties

- Used occasionally
- Share with friends
- Do not share with strangers
**Motivation: Social Inefficiency**

<table>
<thead>
<tr>
<th>Efficiency of Purchase</th>
<th>High (share with many)</th>
<th>Low (share with few)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Likelihood of Purchase</td>
<td>can be <strong>Low</strong> (likely to borrow)</td>
<td>can be <strong>High</strong> (likely to buy)</td>
</tr>
</tbody>
</table>

Q1 “How large can social inefficiency be?”
Q2 “How to nudge people towards efficiency?”
Roadmap

- T1. Structure Analysis (Part 1)
- T2. Anomaly Detection (Part 2)
- T3. Behavior Modeling (Part 3)
  - T3-1. Modeling Purchases (§14)
    - Toy Example <<
    - Game-theoretic Model
    - Best Rental-fee Search
  - ...

- Future Directions
- Conclusions
Social Network

• Consider a **social network**, which is a graph
  ◦ **Nodes**: people
  ◦ **Edges**: friendship

“How many people should buy an IKEA toolkit for everyone to use it?”
Socially Optimal Decision

- The answer is at least 2

- Socially optimal:
  - everyone uses a toolkit
  - with minimum purchases (or with minimum cost)

"Does everyone want to stick to their current decisions?"
Individually Optimal Decision

• The answer is **No**

• **Individually optimal:**
  ◦ everyone best responses to others’ decisions

• **Socially inefficient** (suboptimal):
  ◦ 4 purchases happen when 2 are optimal
Social Inefficiency

- Individually optimal outcome with 6 purchases

“How can we prevent this social inefficiency?”
Moving toward Social Optimum

• Recall the **socially optimal outcome**

“How can we make people stick with this socially optimal outcome?”

“How can we make people stick with this socially optimal outcome?”
**Imposing Rental Fee**

- Renters pay rental fee for getting **permanent access**
- Rental fee is **half** the price of a toolkit

“Does everyone want to stick to their current decisions?”
Socially & Individually Optimal

• The answer is Yes
  ◦ Alice & Bob: are paid more than the price
  ◦ The others: renting is cheaper than buying

• Individually optimal

• Socially optimal with minimum (2) purchases
Socially & Individually Optimal

• The answer is Yes
  ◦ Alice & Bob: are paid more than the price
  ◦ The others: renting is cheaper than buying

Q1 “How do rental fees affect social inefficiency?”
Q2 “What are the ‘socially optimal’ rental fees?”
Roadmap

• T1. Structure Analysis (Part 1)
• T2. Anomaly Detection (Part 2)
• T3. Behavior Modeling (Part 3)
  ° T3-1. Modeling Purchases (§14)
    ▪ Toy Example
    ▪ Game-theoretic Model <<
    ▪ Best Rental-fee Algorithm
  ° ...

• Future Directions
• Conclusions
Formal Game-theoretic Model

- **Players**: nodes in a social network

- **Strategies**:
  - buy a good / rent a good from a friend with a good

- **Nash Equilibrium (NE)**: individually optimal outcome

- **Social Optimum**: socially optimal outcome

- **Inefficiency** of an NE:
  \[
  \frac{\text{# purchases in the NE}}{\text{# purchases in a social optimum}}
  \]
Inefficiency without Rental Fee

• [THM 1] **Existence of NEs**
  - In *every* social network, there exists an NE.

• [THM 2] **Inefficiency without Rental Fee**
  - There exists a social network with $n$ nodes
  - where *all* NEs have $\Theta(n)$ inefficiency.

---

**Input graph**

**Social optimum**

(2 owners)

**Best NE**

(n/2 owners)
Inefficiency with Rental Fee

- [THM 3] Inefficiency with Rental Fee
  - In every social network,
  - if \( \frac{\text{price}}{3} < \text{rental fee} < \text{price} \),
  - then, there exists a socially optimal NE,
  - otherwise ...

Input graph

Social optimum (2 owners)

\( \frac{n - 1}{2} \)

\( \rightarrow \) NE with proper rental fee
Roadmap

- T1. Structure Analysis (Part 1)
- T2. Anomaly Detection (Part 2)
- T3. Behavior Modeling (Part 3)
  - T3-1. Modeling Purchases (§14) <<
    - Toy Example
    - Game-theoretic Model
    - Best Rental-fee Search <<
  - ...

- Future Directions
- Conclusions
Finding Best Rental Fee

• **Given:**
  ◦ a social network
  ◦ a sharable good with price \( p \)

• **Find:** a range of rental fees

• **To Minimize:** inefficiencies of NEs
Searching NEs (SGG-Nash)

- **Linear-time** algorithm for searching NEs

randomly initialize strategies

repeat until an NE is reached
  - for each node
    - optimize its strategy while fixing the others’

- Gives different NEs depending on initial strategies

[THM 4] Convergence
In *every* social network, an NE is reached within 3 iterations.
Scalability of SGG-Nash

- SGG-Nash is linear in the number of edges

Dataset: orkut
Best Rental Fee in Real Graphs

- **Datasets:** [Advogato](#), [Facebook](#), [Google](#)
- **Inefficiency is minimized consistently when**

\[
\frac{\text{price}}{3} < \text{rental fee} < \frac{\text{price}}{2}
\]
Summary of §14

- **Game-theoretic Model:** Sharable good game
- **Theoretical Analysis:**
  - Existence of NEs
  - Inefficiency of NEs
- **Algorithm:** for linear-time NE search
- **Suggestion:** “socially optimal” rental fees
Contributions and Impact (Part 3)

Tools for purchase modeling [IJCAI17]
- Suggest ‘socially optimal’ rental fees
- Media: NewScientist

Tools for progression modeling [WWW18]
- Scale to datasets with a trillion records
- Real-world usage: LinkedIn
## Organization of the Thesis (Recall)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Triangle Count (§§ 3-6)</td>
<td>Anomalous Subgraph (§ 9)</td>
<td>Purchase Behavior (§ 14)</td>
</tr>
<tr>
<td>Summarization (§ 7)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Tensors</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summarization (§ 8)</td>
<td>Dense Subtensors (§§ 10-13)</td>
<td>Progression (§ 15)</td>
</tr>
</tbody>
</table>
Roadmap

• T1. Structure Analysis (Part 1)
• T2. Anomaly Detection (Part 2)
• T3. Behavior Modeling (Part 3)

• Future Directions <<
• Conclusions
Vision: Algorithms for “Big Data”

Platform & System

Scalable Algorithms for Big Data Mining

Applications

Model & Theory
D1: Distributed Graph Stream Processing

“How to analyze large dynamic graphs on a cluster of machines?”

Current
Triangle Counting (§5)

Short Term
More Graph Mining Tasks

Mid Term
Programming Model (Generalization)

Long Term
System / Platform

Sources

Destination
D2: Detecting Adversarial Anomalies

Anomalies / Fraudsters

Detection System

Avoid
Improve

Current

Algorithms for Static Anomalies

Short Term

Profits of Anomalies

Mid Term

Cost to Avoid Algorithms

Long Term

Algorithms Costly to Avoid

Mining Large Dynamic Graphs and Tensors (by Kijung Shin)
D3: Co-Evolution of Beliefs and Graphs

“How to model the co-evolution of nodes’ beliefs and edges?”

Change of **Beliefs**  \hspace{1cm} OR \hspace{1cm} Change of **Edges**

Current

Regression Model [PKDD18]

Short Term

Game Theory / Nash Equilibrium

Mid Term

Prediction Algorithms

Long Term

Reducing Polarization
Roadmap

• T1. Structure Analysis (Part 1)
• T2. Anomaly Detection (Part 2)
• T3. Behavior Modeling (Part 3)
  ◦ T3-1. Modeling Purchases (§14) <<
  ◦ T3-2. Modeling Progression (§ 15) (Skip)

• Future Directions

• Conclusions <<
Conclusion

• **Goal:** “To fully understand and utilize *large dynamic graphs and tensors*”

• **Contributions:** developing *scalable algorithms* for

  T1. Structure Analysis (Part 1)  
  T2. Anomaly Detection (Part 2)  
  T3. Behavior Modeling (Part 3)

• **Impact:**
  - MIT Massachusetts Institute of Technology
  - NAVER
  - NewScientist
  - LinkedIn
  - github.com/kijungs
References


References (cont.)


Thank You!

• Sponsors:

• Admins:

• Collaborators:
Thank You!

- Homepage (Software & Datasets): http://www.cs.cmu.edu/~kijungs/defense/