Mining Billion-Scale Graphs:
Patterns and Algorithms

Christos Faloutsos and U Kang
CMU
Part 2: Algorithms

Complementary to tutorial: Mining Billion-Scale Graphs: Systems and Implementations: Haixun Wang et al
Part 2: Algorithms
Outline

• Problem#1: Patterns in graphs
• Problem#2: Tools
• Problem#3: Scalability - PEGASUS
  – Structure Analysis
  – Eigensolver
  – Graph Layout and Compression
• Conclusions
Our goal:

Open source system for mining huge graphs:

PEGASUS project (PEta GrAph mining System)

- www.cs.cmu.edu/~pegasus
- code and papers
Scalability Challenge

- The sizes of graphs are growing!

Facebook:
- 0.5 billion users
- 60 TBytes/day
- 15 PBytes/total
[Thusoo+ '10]

Bing:
- ClickStream Data
- 0.26 PBytes
- 1 billion query-URL
[Liu+ '09]

Yahoo!
- 1.4 billion web pages
- 6.6 billion edges
[Broder+ '04]

Google:
- 20 PBytes/day
[Dean+ '08]
Scalability Challenge

- The sizes of graphs are growing!

Q: How can we handle large graphs which don’t fit into the memory, or disks of a single machine?

A: Parallelism, with MapReduce!

[Broder+ ’04] [Dean+ ’08]
Background: MapReduce

- MapReduce/Hadoop Framework

HDFS: fault tolerant, scalable, distributed storage system

Mapper: read data from HDFS, output (k,v) pair

Output sorted by the key

Reducer: read output from mappers, output a new (k,v) pair to HDFS
Background: MapReduce

- MapReduce/Hadoop Framework

HDFS: fault tolerant, scalable, distributed storage system

Mapper: read data from HDFS, output (k,v) pair

Output sorted by the key

Reducer: read output from mappers, output a new (k,v) pair to HDFS

Programmers need to provide only map() and reduce() functions
Outline

• Problem#1: Patterns in graphs
• Problem#2: Tools
• Problem#3: Scalability - PEGASUS
  – Structure Analysis
  – Eigensolver
  – Graph Layout and Compression
• Conclusions
Structure Analysis

- How to scale-up structure analysis algorithm?
  - Q1: How to unify many structure analysis algorithms (connected components, PageRank, diameter/radius)?
  - Q2: How to design a scalable algorithm for the structure analysis?
Q1: Unifying Algorithms

- Given a graph, can we compute
  - connected components,
  - PageRank,
  - Random Walk with Restart,
  - diameter/radius
  with *one algorithm*?
Q1: Unifying Algorithms

- Given a graph, can we compute
  - connected components,
  - PageRank,
  - Random Walk with Restart,
  - diameter/radius
with one algorithm?

Yes!

How?
Main Idea

- GIM-V
  - Generalized Iterative Matrix-Vector Multiplication
  - Extension of plain matrix-vector multiplication
  - includes
    - Connected Components
    - PageRank
    - RWR (Random Walk With Restart)
    - Diameter Estimation
Main Idea: Intuition

- Plain M-V multiplication

  - Weighted Combination of Colors
  - $\sim$ Message Passing
Main Idea: Intuition

- Plain M-V multiplication

  - Weighted Combination of Colors
  - $\sim$ Message Passing

$$M = \begin{bmatrix}
A & B & C & D \\
A & 1 & 0 & 0 \\
B & 0 & 1 & 0 \\
C & 0 & 0 & 0.1 \\
D & 1 & 1 & 0.1 \\
\end{bmatrix}$$

$$v = \begin{bmatrix}
\text{Red} \\
\text{Blue} \\
\text{Green} \\
\text{Yellow} \\
\end{bmatrix}$$

$$v' = \sum_{i=1}^{4} m_{4i} v_i$$
Main Idea: Intuition

- Plain M-V multiplication

\[
M = \begin{bmatrix}
A & B & C & D \\
1 & 1 & 0.1 & \\
1 & & & \\
1 & & 0.1 & \\
1 & 1 & 0.1 & \\
\end{bmatrix}
\]

\[v = \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

\[v' = \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

\[
M \times v = v'
\]

\[
v_j' = \sum_{i=1}^{4} m_{ji}v_i
\]
Main Idea: Intuition

- Plain M-V multiplication

\[
M \times v = v'
\]

Three Implicit Operations here:
- multiply \( m_{ji} \) and \( v_i \)
- sum \( n \) multiplication results
- update \( v_j' \)

\[
v_j' = \sum_{i=1}^{4} m_{ji}v_i
\]
Main Idea

- GIM-V
  - Generalizing the three operations leads to many algorithms

<table>
<thead>
<tr>
<th>operations</th>
<th>Standard MV</th>
<th>Con. Cmpt.</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>combine2</td>
<td>Multiply</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>combineAll</td>
<td>Sum</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assign</td>
<td>Assign</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Main Idea

- **GIM-V for Connected Components**

<table>
<thead>
<tr>
<th>operations</th>
<th>Standard MV</th>
<th>Con. Cmpt.</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>combine2</td>
<td>Multiply</td>
<td>Bool. X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>combineAll</td>
<td>Sum</td>
<td>MIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assign</td>
<td>Assign</td>
<td>MIN</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Main Idea

- GIM-V for Connected Components
  - How many connected components?
  - Which node belong to which component?
Main Idea

- GIM-V for Connected Components
Main Idea

- GIM-V for Connected Components

\[ \text{combine}_2(m_{i,j}, v_j) = m_{i,j} \times v_j \]

\[ \text{combine}_\text{All}(x_1, \ldots, x_n) = \min\{x_i \mid i = 1..n\} \]

\[ \text{assign}(v_i, v_{\text{new}}) = \min(v_i, v_{\text{new}}) \]

“Sending Invitations”

“Accept the Smallest”

1 2 3 4 5 6 7 8

\[ \times G \]

1 1 1 1 1 1 1 1
2 1 1 1 1 1 1 1
3 1 1 1 1 1 1 1
4 1 1 1 1 1 1 1
5 1 1 1 1 1 1 1
6 1 1 1 1 1 1 1
7 1 1 1 1 1 1 1
8 1 1 1 1 1 1 1

SIGMOD’12  Faloutsos and Kang (CMU)
Main Idea

- **GIM-V for Connected Components**
  
  \[
  \text{combine}_2(m_{i,j}, v_j) = m_{i,j} \times v_j
  \]
  
  \[
  \text{combine}_\text{All}(x_1, \ldots, x_n) = \min\{x_i | i = 1..n\}
  \]
  
  \[
  \text{assign}(v_i, v_{\text{new}}) = \min(v_i, v_{\text{new}})
  \]

  "Sending Invitations"

  "Accept the Smallest"

  1 GIM-V with MIN() operation
  = find minimum node ids within 1 hop
Main Idea

- **GIM-V for Connected Components**

\[
\text{combine}_2(m_{i,j}, v_j) = m_{i,j} \times v_j
\]

\[
\text{combine}_\text{All}(x_1, \ldots, x_n) = \min \{ x_i \mid i = 1..n \}
\]

\[
\text{assign}(v_i, v_{\text{new}}) = \min(v_i, v_{\text{new}})
\]

“Sending Invitations”

“Accept the Smallest”

(k) GIM-V with MIN() operation

= find minimum node ids within (k) hops

1 2 3 4 5 6 7 8

1 1
2 1 1
3 1 1
4 1
5 1

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & & & & & & & & \\
2 & 1 & 1 & & & & & & \\
3 & 1 & 1 & & & & & & \\
4 & & & 1 & & & & & \\
5 & & & & & 1 & & & \\
\end{array}
\]

\[
\times G = \begin{array}{c}
\min(1, \min(2)) \\
\min(2, \min(1,3)) \\
\min(3, \min(2,4)) \\
\min(4, \min(3)) \\
\min(5, \min(6)) \\
\end{array}
\]

\[
= \begin{array}{c}
1 \\
1 \\
1 \\
1 \\
1 \\
\end{array}
\]

\[
\Rightarrow \begin{array}{c}
1 \\
2 \\
3 \\
2 \\
5 \\
\end{array}
\]

\[
\Rightarrow \begin{array}{c}
1 \\
1 \\
1 \\
1 \\
5 \\
\end{array}
\]

\[
\Rightarrow \begin{array}{c}
1 \\
1 \\
1 \\
1 \\
5 \\
\end{array}
\]

\[
\Rightarrow \begin{array}{c}
5 \\
5 \\
5 \\
5 \\
5 \\
\end{array}
\]

\[
\Rightarrow \begin{array}{c}
5 \\
7 \\
7 \\
7 \\
7 \\
\end{array}
\]
Main Idea

- **GIM-V for Connected Components**

  $\text{combine}2(m_{i,j}, v_j) = m_{i,j} \times v_j$

  $\text{combineAll}(x_1, ..., x_n) = \min \{ x_i | i = 1..n \}  \quad \text{“Sending Invitations”}$

  $\text{assign}(v_i, v_{new}) = \min(v_i, v_{new})  \quad \text{“Accept the Smallest”}$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ \times G \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ = \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Maximum # of iterations : diameter
Main Idea

- GIM-V for PageRank

<table>
<thead>
<tr>
<th>Operations</th>
<th>Standard MV</th>
<th>Con. Cmpt.</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>combine2</td>
<td>Multiply</td>
<td>Multiply</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>combineAll</td>
<td>Sum</td>
<td>MIN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assign</td>
<td>Assign</td>
<td>MIN</td>
<td>Assign</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiply with c
Sum with rj prob
Assign
Main Idea

- GIM-V: PageRank
  - PageRank vector $p$: eigenvector of $A$:
    $1p = A \times p$
    
    
    Where

    
    
    
    $A = cE^T + (1 - c)U$

    Adjacency Matrix
    All elements set to $1/n$
Main Idea

- GIM-V: PageRank
  - Algorithm: Power method
    (many multiplications)

\[
p_{\text{next}} \leftarrow A \times p_{\text{cur}}
\]
Main Idea

- GIM-V: PageRank

\[ p_{next} \leftarrow E^T \times_G p_{cur} \]

Algorithm: Power method
(many multiplications)

\[
\begin{align*}
\text{combine2}(m_{i,j}, v_j) &= c \times m_{i,j} \times v_j \\
\text{combineAll}(x_1, \ldots, x_n) &= \frac{1-c}{n} + \sum x_i \\
\text{assign}(v_i, v_{new}) &= v_{new}
\end{align*}
\]
## Main Idea

### GIM-V

<table>
<thead>
<tr>
<th>Operations</th>
<th>Standard MV</th>
<th>Con. Cmpt.</th>
<th>PageRank</th>
<th>RWR</th>
<th>Diameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>combine2</strong></td>
<td>Multiply</td>
<td>Multiply</td>
<td>Multiply with c</td>
<td>Multiply with c</td>
<td>Multiply bit-vector</td>
</tr>
<tr>
<td><strong>combineAll</strong></td>
<td>Sum</td>
<td>MIN</td>
<td>Sum with rj prob.</td>
<td>Sum with res tart prob</td>
<td>BIT-OR()</td>
</tr>
<tr>
<td><strong>assign</strong></td>
<td>Assign</td>
<td>MIN</td>
<td>Assign</td>
<td>Assign</td>
<td>BIT-OR()</td>
</tr>
</tbody>
</table>
Two Restrictions on HDFS

- [R1] HDFS is location transparent
  - Users don’t know which file is located in which machine

- [R2] A line is never split
  - A large file is split into pieces of a size (e.g. 256 MB)
  - Users don’t know the point of the split
Q2: Fast Algorithms for GIM-V

Given the two restrictions R1 and R2, how can we make faster algorithms for GIM-V in Hadoop?

Three main ideas:
- I1) Block Multiplication
- I2) Clustering
- I3) Compression
Main Idea

I1) Block-Method

\[
\begin{array}{c}
\begin{array}{c}
1 \\
1 \\
1 \\
\end{array} \\
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
\end{array} \\
\begin{array}{c}
5 \\
6 \\
7 \\
8 \\
\end{array} \\
\end{array}
\times G
= 
\begin{array}{c}
\begin{array}{c}
1 \\
1 \\
1 \\
\end{array} \\
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
\end{array} \\
\begin{array}{c}
1 \\
1 \\
1 \\
\end{array} \\
\end{array}
\times G
+ 
\begin{array}{c}
\begin{array}{c}
1 \\
1 \\
1 \\
\end{array} \\
\begin{array}{c}
1 \\
2 \\
3 \\
4 \\
\end{array} \\
\begin{array}{c}
1 \\
1 \\
8 \\
\end{array} \\
\end{array}
\times G
\end{array}
\]
Main Idea

I1) Block-Method

1. Group elements together into 1 line
2. Storage for an element: \(2\log n\) bits -> \(2\log b\) bits
3. Adjust the MapReduce code (block multiplication)
Main Idea

- I1) Block-Method

Thanks to the encoding,
- file size is decreased,
- shuffle time is decreased.
Main Idea

Q: Can we do even better?
Main Idea

- I2) Clustering

A: preprocessing for clustering (only green blocks are stored in HDFS)
Main Idea

- I3) Compression

A: compress clustered blocks
Result

### Block Encoding

<table>
<thead>
<tr>
<th></th>
<th>Compression</th>
<th>Clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>NNB</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NCB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CCB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

### File Size (MB)

- YahooWeb: 5x smaller
- Twitter Graph: 5x smaller
- Random: 43x smaller

### Running Time in Seconds

- YahooWeb: 2x faster
- Twitter Graph: 1.7x faster
- Random: 9.2x faster
A: Proposed Method (CCB) provides 43x smaller storage, 9.2x faster running time
Outline

• Problem#1: Patterns in graphs
• Problem#2: Tools
• Problem#3: Scalability - PEGASUS
  – Structure Analysis
  – Eigensolver
  – Graph Layout and Compression
• Conclusions
Background: Eigensolver

- Eigensolver
  - Given: (adjacency) matrix $A$,
  - Compute: top $k$ eigenvalues and eigenvectors of $A$
  - Application:
    - SVD
    - triangle counting
    - spectral clustering
Problem Definition

Q4: How to design a billion-scale eigensolver?
- Existing eigensolver: can handle millions of nodes and edges
Efficient Eigensolver

- Lanczos Iterations

\[ \beta_0 = 0, \quad q_0 = 0, \quad b = \text{arbitrary}, \quad q_1 = b/\|b\| \]

for \( n = 1, 2, 3, \ldots \)

\[ v = Aq_n \]
\[ \alpha_n = q_n^T v \]
\[ v = v - \beta_{n-1}q_{n-1} - \alpha_n q_n \]
\[ \beta_n = \|v\| \]
\[ q_{n+1} = v/\beta_n \]

1 matrix-vector multiplication per iteration
Proposed Method

- HEigen algorithm (Hadoop Eigen-solver)
  - Selectively parallelize ‘Lanczos-SO’ algorithm
  - Block encoding
  - Exploiting skewness in matrix-matrix mult.
    - \((m >> n > k)\)
Skewed Matrix-Matrix Mult.

- Multiply $Q_n^{mxn}$ and $H^{nxk}$ (m >> n > k)

  - Naïve multiplication: too expensive
  - Proposed:
    - `cache’-based multiplication: broadcast the small matrix $x H$ to all the machines that contains $Q_n$

$Q_n$: O(100 Gbytes)
$H$: O(Kbytes)
Skewed Matrix-Matrix Mult.

- `cache'-based multiplication: broadcast the small matrix $H$ to all the machines that contains $Q_n$

```
      HDFS

  $Q_n^{(1/3)}$  $Q_n^{(2/3)}$  $Q_n^{(3/3)}$

  Map 0  Map 1  Map 2

  Shuffle

  Reduce 0  Reduce 1
```

$H \times A's$'s Eigenvector

$Q_n^{(1/3)} \approx \approx \approx \approx \approx \approx$

$H \times Q_n \times H$

$Q_n^{(1/3)} \times n \times k$

$Q_n \times m \times n$

$H \times m \times k$
Skewed Matrix-Matrix Mult.

Which Matrix-Matrix multiplication algorithm runs the fastest?

- MM: naïve mat-mat mult.
- IMV: naïve iterative mat-vec mult.
- CBMV: cache-based iterative mat-vec mult.
- CBMM: cache-based mat-mat mult.

Cache-based MM runs 76x faster
Outline

• Problem#1: Patterns in graphs
• Problem#2: Tools
• Problem#3: Scalability - PEGASUS
  – Structure Analysis
  – Eigensolver
  – Graph Layout and Compression
• Conclusions
Node Order Matters

- A graph and the adjacency matrix

![Graph and Adjacency Matrix](image-url)
Node Order Matters

- Same graphs with different orderings
Good ordering = Good compression

- Same graphs with different orderings

Many sparse blocks

Few dense blocks
Application

- Block-based matrix-vector multiplication

Adjacency Matrix

\[
\begin{bmatrix}
M_0 & M_1 & M_2 \\
M_3 & M_4 & M_5 \\
M_6 & M_7 & M_8
\end{bmatrix}
\times
\begin{bmatrix}
v_0 \\
v_1 \\
v_2
\end{bmatrix}
= \begin{bmatrix}
M_0 \\
M_3 \\
M_6
\end{bmatrix}
\times
\begin{bmatrix}
v_0 \\
v_1 \\
v_2
\end{bmatrix}
+ \begin{bmatrix}
M_1 \\
M_4 \\
M_7
\end{bmatrix}
\times
\begin{bmatrix}
v_0 \\
v_1 \\
v_2
\end{bmatrix}
+ \begin{bmatrix}
M_2 \\
M_5 \\
M_8
\end{bmatrix}
\times
\begin{bmatrix}
v_0 \\
v_1 \\
v_2
\end{bmatrix}
= \begin{bmatrix}
w_0 \\
w_1 \\
w_2
\end{bmatrix}
\]

Few, dense blocks

=> Better compression, faster running time
Problem Definition

- Given a graph, how can we lay-out its edges so that nonzero elements are well-clustered?
- Better clustering = better compression

Many sparse blocks

Few dense blocks

SIGMOD’12
Main Result

Original

SlashBurn

SIGMOD’12  Faloutsos and Kang (CMU)
Outline

• …

• Problem#3: Scalability - PEGASUS
  – Structure Analysis
  – Eigensolver
  – Graph Layout and Compression
    • Proposed Method
    • Results

• Conclusions
Survey

Given a graph, how can we lay-out its edges so that nonzero elements are well-clustered?

1) Graph based clustering
   - Normalized cut, spectral clustering

2) Heuristics
   - Lexicographic ordering for Web
   - Shingle ordering
1) Graph Based Clustering

- Goal: find homogeneous sets of nodes from graphs
  - E.g.) Spectral clustering and normalized cut
  - Many intra-edges, few inter-edges

Caveman Communities
1) Graph Based Clustering

- Goal: find homogeneous sets of nodes from graphs
  - E.g.) Spectral clustering and normalized cut
  - Many intra-edges, few inter-edges

Caveman Communities
1) Graph Based Clustering

- But, real graphs: no good cuts
  - [Tauro+ 01], [Siganos+ 06], [Chakrabarti +04], [Lesko vec+ 08]
1) Graph Based Clustering

- But, real graphs: no good cuts
  - [Tauro+ 01], [Siganos+ 06], [Chakrabarti +04], [Lesko vec+ 08]
1) Graph Based Clustering

- But, real graphs: no good cuts
  - [Tauro+ 01], [Siganos+ 06], [Chakrabarti +04], [Lesko vec+ 08]

- What should we do?

- What should we do?
2) Older Heuristics

- **Web graph: lexicographic ordering [Boldi+, 04]**
  - Locality: many intra edges between neighbors
  - Similarity: out links of neighbors are similar

- **Social network: shingle ordering [Chierichetti+, 09]**
  - Group nodes with similar our-neighbors
Summary: Previous Work

- Tries to find homogenous regions for graph compression
  - Fails to find them, because they often don’t exist.

[Diagram of graph structures]
Our Observation

- Caveman assumption
Our Observation

- Caveman assumption: wrong!
Our Solution

- Caveman assumption: wrong!

- Instead, we envision graphs as nodes connected by connectors connected by super connectors…
Our Solution

- Instead, we envision graphs as **nodes** connected by **connectors** connected by **super connectors** …

- Use “Graph Shattering” to `peel’ the graphs from **super connectors**, and then **connectors**, …
Graph Shattering

- $k$-shattering of a graph $G$
  - Removes top $k$ connectors and their incident edges from $G$
Graph Shattering

Before shattering
Graph Shattering

Before shattering

After shattering
Graph Shattering

Observations in real graphs

O1. Portion of GCC is much smaller after shattering

O2. A lot of disconnected components
Slash-Burn method (intuition)

- ‘burn’ the top $k$ connectors, and ‘slash’ the edges
- Move $k$ connectors to the front of the row/column, sort connected components by decr. size
- Recurse on the remaining GCC
Outline

• ...  
• Problem#3: Scalability - PEGASUS  
  – Structure Analysis  
  – Eigensolver  
  – Graph Layout and Compression  
    • Proposed Method  
    • Results  
• Conclusions
Goal of Experiments

- [Q1] Compression savings?
- [Q2] Running time savings?
A1. Compression

![Graphs showing compression results for different datasets and methods.]

SIGMOD’12

Faloutsos and Kang (CMU)
A1. Compression

- Cost functions used
  1) Number of non-empty blocks
  2) Information theoretic cost: minimum bits to encode non-zero elements inside blocks

\[
|T| \cdot 2 \log \frac{n}{b} + \sum_{\tau \in T} b^2 \cdot H\left(\frac{\tau}{b^2}\right)
\]

- Model complexity costs given the model

- \( |T| \): # of nonempty blocks
- \( n \): # of nodes
- \( b \): block width
- \( H() \): Shannon entropy func
A1. Compression

Cost comparison

- SlashBurn outperforms all competitors for all dataset!
  (smallest number of nonempty blocks, as well as bits per edge)
A2. Running Time

- SlashBurn outperforms all competitors! (running time as well as file size)
Outline

• Introduction – Motivation
• Problem#1: Patterns in graphs
• Problem#2: Tools
• Problem#3: Scalability - PEGASUS
• Conclusions
Conclusions

- **PEGASUS**: Peta-Scale Graph Mining System
  - Patterns and anomalies in large graphs
    - PageRank, Connected Components, Radius, Eigensolver
  - Outreach
    - Microsoft: part of Hadoop distribution for Windows Azure
    - One of the core systems for several DARPA projects (ADA MS, INARC, DTRA)

[www.cs.cmu.edu/~pegasus]
Conclusions

- High impact applications require large graph mining

Cyber Security  Fraud Detection  Social Network  Search Engine  Health Care
Thank you!

Complementary tutorial: *Mining Billion-Scale Graphs: Systems and Implementations*: Haixun Wang et al