GRAPH COMPRESSION AND SUMMARIZATION

Wei Zhang
Dept. of Information Engineering
The Chinese University of Hong Kong
Most of the slides are borrowed from the authors’ original presentation.

- [http://videolectures.net/kdd09_kumar_ocsn/](http://videolectures.net/kdd09_kumar_ocsn/)
GRAPH SUMMARIZATION WITH BOUNDED ERROR

- Saket Navlakha (UMCP)
- Rajeev Rastogi (Yahoo! Labs, India)
- Nisheeth Shrivastava (Bell Labs India)
Many interactions can be represented as graphs

- Webgraphs: search engine, etc.
- Netflow graphs (which IPs talk to each other): traffic patterns, security, worm attacks
- Social (friendship) networks: mine user communities, viral marketing
- Email exchanges: security, virus spread, spam detection
- Market basket data: customer profiles, targeted advertizing

Need to compress, understand

- Webgraph ~ 50 billion edges; social networks ~ few million, growing quickly
- Compression reduces size to one-tenth (webgraphs)
OUR APPROACH

- Graph Compression (reference encoding)
  - Not applicable to all graphs: use urls, node labels for compression
  - Resulting structure is hard to visualize/interpret

- Graph Clustering
  - Nice summary, works for generic graphs
  - No compression: needs the same memory to store the graph itself

- Our MDL-based representation $R = (S,C)$
  - *S is a high-level summary graph*: compact, highlights dominant trends, easy to visualize
  - *C is a set of edge corrections*: help in reconstructing the graph
  - Compression based on MDL principle: minimize cost of $S+C$
    - Information-theoretic approach; parameter less; applicable to any graph
  - Novel Approximate Representation: reconstructs graph with bounded error ($\epsilon$); results in better compression
**How do we compress?**

- Compression possible (S)
  - Many nodes with similar neighborhoods
    - Communities in social networks; link-copying in webpages
  - Collapse such nodes into *supernodes* (clusters) and the edges into *superedges*
    - Bipartite subgraph to two supernodes and a superedge
    - Clique to supernode with a “self-edge”
HOW DO WE COMPRESS?

- Compression possible (S)
  - Many nodes with similar neighborhoods
    - Communities in social networks; link-copying in webpages
    - Collapse such nodes into supernodes (clusters) and the edges into superedges
      - Bipartite subgraph to two supernodes and a superedge
      - Clique to supernode with a “self-edge”

- Need to correct mistakes (C)
  - Most superedges are not complete
    - Nodes don’t have exact same neighbors: friends in social networks
  - Remember edge-corrections
    - Edges not present in superedges (-ve corrections)
    - Extra edges not counted in superedges (+ve corrections)

- Minimize overall storage cost = S+C

Cost = 14 edges

![Graph diagram]

$X = \{d,e,f,g\}$

$Y = \{a,b,c\}$

Correction

$(a,h)$
$(c,i)$
$(c,j)$
$-(a,d)$

Cost = 5
(1 superedge + 4 corrections)
Reprentation Structure $R=(S,C)$

- Summary $S(V_S, E_S)$
  - Each supernode $v$ represents a set of nodes $A_v$
  - Each superedge $(u,v)$ represents all pair of edges $\pi_{uv} = A_u \times A_v$

- Corrections $C$: $\{(a,b); a$ and $b$ are nodes of $G\}$

- Supernodes are key, superedges/corrections easy
  - $A_{uv}$ actual edges of $G$ between $A_u$ and $A_v$
  - Cost with $(u,v) = 1 + |\pi_{uv} - E_{uv}|$
  - Cost without $(u,v) = |E_{uv}|$
  - Choose the minimum, decides whether edge $(u,v)$ is in $S$
**Representation Structure** \( R=(S, C) \)

- **Summary** \( S(V_s, E_s) \)
  - Each supernode \( v \) represents a set of nodes \( A_v \)
  - Each superedge \( (u,v) \) represents all pair of edges \( \pi_{uv} = A_u \times A_v \)

- **Corrections** \( C : \{ (a,b); \ a \text{ and } b \text{ are nodes of } G \} \)

- **Supernodes** are key, superedges/corrections easy
  - \( A_{uv} \) actual edges of \( G \) between \( A_u \) and \( A_v \)
  - Cost with \( (u,v) = 1 + |\pi_{uv} - E_{uv}| \)
  - Cost without \( (u,v) = |E_{uv}| \)
  - Choose the minimum, decides whether edge \( (u,v) \) is in \( S \)

- **Reconstructing the graph from** \( R \)
  - For all superedges \( (u,v) \) in \( S \), insert all pair of edges \( \pi_{uv} \)
  - For all +ve corrections \( +(a,b) \), insert edge \( (a,b) \)
  - For all -ve corrections \( -(a,b) \), delete edge \( (a,b) \)
**Representation Structure** \( R = (S, C) \)

- **Summary** \( S(V_S, E_S) \)
  - Each supernode \( v \) represents a set of nodes \( A_v \)
  - Each superedge \( (u,v) \) represents all pair of edges \( \pi_{uv} = A_u \times A_v \)

- ** Corrections** \( C : \{(a,b); \ a \text{ and } b \text{ are nodes of } G\} \)

- **Supernodes are key, superedges/corrections easy**
  - \( A_{uv} \) actual edges of \( G \) between \( A_u \) and \( A_v \)
  - Cost with \( (u,v) = 1 + |\pi_{uv} - E_{uv}| \)
  - Cost without \( (u,v) = |E_{uv}| \)
  - Choose the minimum, decides whether edge \( (u,v) \) is in \( S \)

- **Reconstructing the graph from \( R \)**
  - For all superedges \( (u,v) \) in \( S \), insert all pair of edges \( \pi_{uv} \)
  - For all +ve corrections \( +(a,b) \), insert edge \( (a,b) \)
  - For all -ve corrections \( -(a,b) \), delete edge \( (a,b) \)
**Representation Structure** $R = (S, C)$

- **Summary** $S(V_S, E_S)$
  - Each supernode $v$ represents a set of nodes $A_v$
  - Each superedge $(u,v)$ represents all pair of edges $\pi_{uv} = A_u \times A_v$

- **Corrections** $C$: $\{(a,b); \ a \text{ and } b \text{ are nodes of } G\}$
- **Supernodes** are key, superedges/corrections easy
  - $A_{uv}$ actual edges of $G$ between $A_u$ and $A_v$
  - Cost with $(u,v) = 1 + |\pi_{uv} - E_{uv}|$
  - Cost without $(u,v) = |E_{uv}|$
  - Choose the minimum, decides whether edge $(u,v)$ is in $S$

- **Reconstructing the graph from** $R$
  - For all superedges $(u,v)$ in $S$, insert all pair of edges $\pi_{uv}$
  - For all +ve corrections $+(a,b)$, insert edge $(a,b)$
  - For all -ve corrections $-(a,b)$, delete edge $(a,b)$
Approximate Representation $R_\varepsilon$

- **Approximate representation**
  - Recreating the input graph *exactly* is not always necessary
  - Reasonable approximation enough: to compute communities, anomalous traffic patterns, etc.
  - Use approximation leeway to get further cost reduction

- **Generic Neighbor Query**
  - Given node $v$, find its neighbors $N_v$ in $G$
  - Apx-nbr set $N'_v$ estimates $N_v$ with $\varepsilon$-accuracy
  - Bounded error: $\text{error}(v) = |N'_v - N_v| + |N_v - N'_v| < \varepsilon$
    - Number of neighbors added or deleted is at most $\varepsilon$-fraction of the true neighbors

- **Intuition for computing $R_\varepsilon$**
  - If correction $(a,d)$ is deleted, it adds error for both $a$ and $d$
  - From exact representation $R$ for $G$, remove (maximum) corrections s.t. $\varepsilon$-error guarantees still hold
COMPARISON WITH EXISTING TECHNIQUES

- **Webgraph compression** [Adler-DCC-01]
  - Use nodes sorted by urls: not applicable to other graphs
  - More focus on bitwise compression: represent sequence of neighbors (ids) using smallest bits

- **Clique stripping** [Feder-pods-99]
  - Collapses edges of complete bi-partite subgraph into single cluster
  - Only compresses very large, complete bi-cliques

- **Representing webgraphs** [Raghavan-icde-03]
  - Represent webgraphs as SNodes, Sedges
  - Use urls of nodes for compression (not applicable for other graphs)
  - No concept of approximate representation
OUTLINE

- Compressed graph
  - MDL representation $R=(S,C)$; $\epsilon$-representation
- Computing $R$
  - GREEDY, RANDOMIZED
- Computing $R_\epsilon$
  - APX-MDL, APX-GREEDY
- Experimental results
- Conclusions and future work
**GREEDY**

- **Cost of merging supernodes** u and v into single supernode w
  - Recall: cost of a superedge (u,x):
    \[ c(u,x) = \min\{|\pi_{vx} - A_{vx}| + 1, |A_{vx}|\} \]
  - \( c_u = \) sum of costs of all its edges = \( \Sigma x c(u,x) \)
  - \( s(u,v) = (c_u + c_v - c_w)/(c_u + c_v) \)

- **Main idea: recursive bottom-up merging of supernodes**
  - If \( s(u,v) > 0 \), merging u and v reduces the cost of reduction
  - Normalize the cost: remove bias towards high degree nodes
  - Making supernodes is the key: superedges and corrections can be computed later

\[ c_u = 5; \quad c_v = 4 \]
\[ c_w = 6 \text{ (3 edges, 3 corrections)} \]
\[ s(u,v) = 3/9 \]
**GREEDY**

- Recall: \( s(u,v) = \frac{c_u + c_v - c_w}{c_u + c_v} \)
- GREEDY algorithm
  - Start with \( S = G \)
  - At every step, pick the pair with max \( s(\cdot) \) value, merge them
  - If no pair has positive \( s(\cdot) \) value, stop

---

![Graph example]

\( s(b,c) = 0.5 \)
\[ c_b = 2; \ c_c = 2; \ c_{bc} = 2 \]

\( s(g,h) = \frac{3}{7} \)
\[ c_g = 3; \ c_h = 4; \ c_{gh} = 4 \]

\( s(e,f) = \frac{1}{3} \)
\[ c_e = 2; \ c_f = 1; \ c_{ef} = 2 \]
RANDOMIZED

- GREEDY is slow
  - Need to find the pair with (globally) max $s(.)$ value
  - Need to process all pair of nodes at a distance of 2-hops
  - Every merge changes costs of all pairs containing $N_w$

- Main idea: light weight randomized procedure
  - Instead of choosing the globally best pair,
  - Choose (randomly) a node $u$
  - Merge the best pair containing $u$
RANDOMIZED

- Randomized algorithm
  - Unfinished set \( U = V_G \)
  - At every step, randomly pick a node \( u \) from \( U \)
  - Find the node \( v \) with max \( s(u,v) \) value
  - If \( s(u,v) > 0 \), then merge \( u \) and \( v \) into \( w \), put \( w \) in \( U \)
  - Else remove \( u \) from \( U \)
  - Repeat till \( U \) is not empty

\[ \text{Picked } e; \ s(e,f)=\frac{3}{5} \]
\[ [ \ c_e = 3; \ c_f=2; \ c_{ef}=3 \ ] \]
OUTLINE

- Compressed graph
  - MDL representation $R=(S,C)$; $\epsilon$-representation
- Computing $R$
  - GREEDY, RANDOMIZED
- Computing $R_\epsilon$
  - APX-MDL, APX-GREEDY
- Experimental results
- Conclusions and future work
**COMPUTING APPROX REPRESENTATION**

- Reducing size of corrections
  - *Correction graph* $H$: For every (+ve or –ve) correction $(a,b)$ in $C$, add edge $(a,b)$ to $H$
  - Removing $(a,b)$ reduces size of $C$, but adds error of 1 to $a$ and $b$
  - Recall bounded error: $\text{error}(v) = |N'_v - N_v| + |N_v - N'_v| < \epsilon |N_v|$
  - Implies in $H$, we can remove up to $b_v = \epsilon |N_v|$ edges incident on $v$
  - Maximum cost reduction: remove subset $M$ of $E_H$ of max size s.t. $M$ has at most $b_v$ edges incident on $v$

- Same as the $b$-matching problem
  - Find the matching $M \subset E_G$ s.t. at most $b_v$ edges incident on $v$ are in $M$
  - For all $b_v = 1$, traditional matching problem
  - Solvable in time $O(mn^2)$ [Gabow-STOC-83] (for graph with $n$ nodes and $m$ edges)
Computing Approximate Representation

- Reducing size of summary
  - Removing superedge \((a,b)\) implies bulk removal of all pair edges \(\pi_{uv}\)
  - But, each node in \(A_u\) and \(A_v\) has different \(b\) value
  - Does not map to a clean matching-type problem

- A greedy approach
  - Pick superedges by increasing \(|\pi_{uv}|\) value
  - Delete \((u,v)\) if that doesn’t violate \(\epsilon\)-bound for nodes in \(A_u \cup A_v\)
  - If there is correction \((a,b)\) for \(\pi_{uv}\) in \(C\), we cannot remove \((u,v)\); since removing \((u,v)\) violates error bound for \(a\) or \(b\)
APXMDL

- Compute the $R(S, C)$ for $G$
- Find $C_ε$
  - Compute $H$, with $V_H = C$
  - Find maximum $b$-matching $M$ for $H$; $C_ε = C - M$
- Find $S_ε$
  - Pick superedges $(u, v)$ in $S$ having no correction in $C_ε$
    in increasing $|π_{uv}|$ value
  - Remove $(u, v)$ if that doesn’t violate $ε$-bound for any node in $A_u \cup A_v$
- Axp-representation $R_ε = (C_ε, S_ε)$
OUTLINE

- Compressed graph
  - MDL representation $R=(S,C)$; $\epsilon$-representation
- Computing $R$
  - GREEDY, RANDOMIZED
- Computing $R_\epsilon$
  - APX-MDL, APX-GREEDY
- Experimental results
- Conclusions and future work
EXPERIMENTAL SET-UP

- **Algorithms to compare**
  - Our techniques GREEDY, RANDOMIZED, APXMDL
  - REF: reference encoding used for web-graph compression
    (we disabled bit-level encoding techniques)
  - GRAC: graph clustering algorithm
    (make supernodes for clusters returned)

- **Datasets**
  - CNR: web-graph dataset
  - Routeview: autonomous systems topology of the internet
  - Wordnet: English words, edges between related words (synonym, similar, etc.)
  - Facebook: social networking
COST REDUCTION (CNR DATASET)

- Reduces the cost down to 40%
- Cost of GREEDY 20% lower than RANDOMIZED
- RANDOMIZED is 60% faster than GREEDY
Comparison with other schemes

Our techniques give much better compression.
80% cost of representation is due to corrections.

Cost Breakup (CNR dataset)

- No. supernodes
- Summary Cost
- Corrections
- Total Cost

Cost (k)

Size (k)
Cost reduces linearly as $\varepsilon$ is increased; With $\varepsilon = .1$, 10% cost reduction over $R$. 
CONCLUSIONS

- MDL-based representation $R(S,C)$ for graphs
  - Compact summary $S$: highlights trends
  - Corrections $C$: reconstructs graph together with $S$
  - Extend to approximate representation with bounded error
  - Our techniques, GREEDY, RANDOMIZED give up to 40% cost reduction

- Future directions
  - Hardness of finding minimum-cost representation
  - Running graph algorithms (approximately) directly on the compressed structure: apx-shortest path with bounded error on $S$?
  - Extend to labeled/weighted edges
On Compressing Social Networks

- Flavio Chierichetti, University of Rome
- Ravi Kumar, Yahoo! Research
- Silvio Lattanzi, University of Rome
- Michael Mitzenmacher, Harvard
- Alessandro Panconesi, University of Rome
- Prabhakar Raghavan, Yahoo! Research
BEHAVIOURAL GRAPHS

- Web graphs
- Host graphs
- Social networks
- Collaboration networks
- Sensor networks
- Biological networks
- ...

Research trends
- Empirical analysis: examining properties of real-world graphs
- Modeling: finding good models for behavioural graphs

There has been a tendency to lump together behavioural graphs arising from a variety of contexts
Properties of Behavioural Graphs

- Power law degree distribution
  - Heavy tail

- Clustering
  - High clustering coefficient

- Communities and dense subgraphs
  - Abundance; locally dense, globally sparse; spectrum

- Connectivity
  - Exhibit a “bow-tie” structure; low diameter; small-world phenomenon: Any two vertices are connected by a short path. Two vertices having a common neighbor are more likely to be neighbors.
A REMARKABLE EMPIRICAL FACT

- Snapshots of the web graph can be compressed using less than 3 bits per edge
  - Boldi, Vigna WWW 2004
- Improved to ~2 bits using another data mining inspired compression technique
  - Buehrer, Chellapilla WSDM 2008
- More recent improvements
  - Boldi, Santinin, Vigna WAW 2009

Key insights
1. Many web pages have similar set of neighbors
2. Edges tend to be “local”
ARE SOCIAL NETWORKS COMPRESSIBLE?

- Review of BV compression
- A different compression mechanism that works better for social networks
- A heuristic
- its performance
- and a formalization
- Why study this question?
  - Efficient storage
    - Serve adjacency queries efficiently in-memory
    - Archival purposes – *multiple snapshots*
  - Obtain insights
    - Compression has to utilize special structure of the network
    - Study the randomness in such networks
ADJACENCY TABLE REPRESENTATION

- Each row corresponds to a node u in the graph
- Entries in a row are sorted integers, representing the neighborhood of u, i.e., edges (u, v)
  - 1: 1, 2, 4, 8, 16, 32, 64
  - 2: 1, 4, 9, 16, 25, 36, 49, 64
  - 3: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144
  - 4: 1, 4, 8, 16, 25, 36, 49, 64
- Can answer adjacency queries fast
- Expensive (better than storing a list of edges)
**BOLDI-VIGNA (BV): MAIN IDEAS**

- **Similar neighborhoods:** The neighborhood of a web page can be expressed in terms of other web pages with similar neighborhoods
  - Rows in adjacency table have similar entries
  - Possible to choose to *prototype* row

- **Locality:** Most edges are intra-host and hence local
  - Small integers can represent edge destination wrt source

- **Gap encoding:** Instead of storing destination of each edge, store the difference from the previous entry in the same row
FINDING SIMILAR NEIGHBORHOODS

- Canonical ordering: Sort URLs lexicographically, treating them as strings

  ...

  17: www.stanford.edu/alchemy
  18: www.stanford.edu/biology
  19: www.stanford.edu/biology/plant
  20: www.stanford.edu/biology/plant/copyright
  21: www.stanford.edu/biology/plant/people
  22: www.stanford.edu/chemistry

  ...

- This gives an identifier for each URL
  Source and destination of edges are likely to get nearby IDs
  - Templated webpages
  - Many edges are intra-host or intra-site
GAP ENCODINGS

- Given a sorted list of integers x, y, z, ..., represent them by x, y-x, z-y, ...
- Compress each integer using a code
  - γ code: x is represented by concatenation of unary representation of $\lfloor \lg x \rfloor$ (length of x in bits) followed by binary representation of $x - 2^{\lfloor \lg x \rfloor}$
    Number of bits = $1 + 2\lfloor \lg x \rfloor$
    (see slide 12, http://vigna.dsi.unimi.it/algoweb/webgraph.pdf)
  - δ code: ...
  - Information theoretic bound: $1 + \lfloor \lg x \rfloor$ bits
  - ζ code: Works well for integers from a power law
  Boldi Vigna DCC 2004
**BV COMPRESSION**

- Each node has a unique ID from the canonical ordering
- Let $w = \text{copying window parameter}$
- To encode a node $v$
  - Check if out-neighbors of $v$ are similar to any of $w-1$ previous nodes in the ordering
  - If yes, let $u$ be the prototype: use $\lg w$ bits to encode the gap from $v$ to $u$ + difference between out-neighbors of $u$ and $v$
  - If no, write $\lg w$ zeros and encode out-neighbors of $v$ explicitly
- Use gap encoding on top of this
MAIN ADVANTAGES OF BV

- Depends only on locality in a canonical ordering
  - Lexicographic ordering works well for web graph
- Adjacency queries can be answered very efficiently
  - To fetch out-neighbors, trace back the chain of prototypes until a list whose encoding begins with \( lg \) zeros is obtained (no-prototype case)
  - This chain is typically short in practice (since similarity is mostly intra-host)
  - Can also explicitly limit the length of the chain during encoding
- Easy to implement and a one-pass algorithm
Social networks are highly *reciprocal*, despite being directed
- If A is a friend of B, then it is likely B is also A’s friend

(u, v) is reciprocal if (v, u) also exists
reciprocal(u) = set of v’s such that (u, v) is reciprocal

How to exploit reciprocity in compression?
- Can avoid storing reciprocal edges twice
- Just the reciprocity “bit” is sufficient
Given a canonical ordering of nodes and copying window \( w \)

To encode a node \( v \)
- Base information: encode out-degree of \( v \) minus 1 (if self loop) minus \#reciprocal(\( v \)) + “self-loop” bit
- Try to choose a prototype \( u \) as in BV within a window \( w \)
- If yes, encode the difference between out-neighbors of \( u \) and non-reciprocal out-neighbors of \( v \)
  - Encode the gap between \( u \) and \( v \)
  - Specify which out-neighbors of \( u \) are present in \( v \)
  - For the rest of out-neighbors of \( v \), encode them as gaps
- Encode the reciprocal out-neighbors of \( v \)
  - For each out-neighbor \( v' \) of \( v \) and \( v' > v \), store if \( v' \in \text{reciprocal}(v) \) or not; discard the edge (\( v' , v \))
CANONICAL ORDERINGS

- BV and BL compressions depend just on obtaining a canonical ordering of nodes
  - This canonical ordering should exploit neighborhood similarity and edge locality
- Question: how to obtain a good canonical ordering?
  - Unlike the web page case, it is unclear if social networks have a natural canonical ordering
- Caveat: BV/BL is only one genre of compression scheme
  - Lack of good canonical ordering does not mean graph is incompressible
SOME CANONICAL ORDERINGS IN BEHAVIORAL GRAPHS

- Random order
- Natural order
  - Time of joining in a social network
  - Lexicographic order of URLs
  - Crawl order
- Graph traversal orders
  - BFS and DFS
- Geographic location: order by zip codes
  - Produces a bucket order
- Ties can be broken using more than one order
## Performance of Simple Orderings

<table>
<thead>
<tr>
<th>Graph</th>
<th>#nodes</th>
<th>#edges</th>
<th>%reciprocal edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>25.1M</td>
<td>69.7M</td>
<td>64.4</td>
</tr>
<tr>
<td>UK host graph</td>
<td>0.58M</td>
<td>12.8M</td>
<td>18.6</td>
</tr>
<tr>
<td>IndoChina</td>
<td>7.4M</td>
<td>194.1M</td>
<td>20.9</td>
</tr>
</tbody>
</table>

### BV

<table>
<thead>
<tr>
<th>Graph</th>
<th>Natural</th>
<th>Random</th>
<th>DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>21.8</td>
<td>23.9</td>
<td>22.9</td>
</tr>
<tr>
<td>UK host</td>
<td>10.8</td>
<td>15.5</td>
<td>14.6</td>
</tr>
<tr>
<td>IndoChina</td>
<td>2.02</td>
<td>21.44</td>
<td>-</td>
</tr>
</tbody>
</table>

### BL

<table>
<thead>
<tr>
<th>Graph</th>
<th>Natural</th>
<th>Random</th>
<th>DFS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>16.4</td>
<td>17.8</td>
<td>17.2</td>
</tr>
<tr>
<td>UK host</td>
<td>10.5</td>
<td>14.5</td>
<td>13.8</td>
</tr>
<tr>
<td>IndoChina</td>
<td>2.35</td>
<td>17.6</td>
<td>-</td>
</tr>
</tbody>
</table>
SHINGLE ORDERING HEURISTIC

- Obtain a canonical ordering by bringing nodes with similar neighborhoods close together.
- Fingerprint neighborhood of each node and order the nodes according to the fingerprint.
  - If fingerprint can capture neighborhood similarity and edge locality, then it will produce good compression via BV/BL, provided the graph has amenable.
- Use Jaccard coefficient to measure similarity between nodes.

\[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} \]
A FINGERPRINT FOR JACCARD

- Fingerprint to measure set overlap
  - Shingles have since seen wide usage to estimate the similarity of web pages using a particular feature extraction scheme based on overlapping windows of terms (motivating the name “shingles”)

\[
M_{\pi}(A) = \pi^{-1}(\min_{a \in A} \{ \pi(a) \})
\]

\[
\Pr_{\pi}[M_{\pi}(A) = M_{\pi}(B)] = \frac{|A \cap B|}{|A \cup B|}
\]

- The probability that the smallest element of \( A \) and \( B \) is the same, where smallest is defined by the permutation \( \pi \), is exactly the similarity of the two sets according to the Jaccard coefficient.

- Min-wise independent permutations suffice
  - Broder, Charikar, Frieze, Mitzenmacher STOC 1998

- Hash functions work well in practice
SHINGLE ORDERING HEURISTIC (CONT'D)

- Fingerprint of a node \( u = M_\pi(\text{out-neighbors of } u) \)
- Order the nodes by their fingerprint
  - Two nodes with lot of overlapping neighbors are likely to have same shingle
- Double shingle order: break ties within shingle order using a second shingle
# Performance of Shingle Ordering

<table>
<thead>
<tr>
<th>Graph</th>
<th>BV Natural</th>
<th>BV Shingle</th>
<th>BV Double shingle</th>
<th>BL Natural</th>
<th>BL Shingle</th>
<th>BL Double shingle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flickr</td>
<td>21.8</td>
<td>13.5</td>
<td>13.5</td>
<td>Flickr</td>
<td>16.4</td>
<td>10.9</td>
</tr>
<tr>
<td>UK host</td>
<td>10.8</td>
<td>8.2</td>
<td>8.1</td>
<td>UK host</td>
<td>10.5</td>
<td>8.2</td>
</tr>
<tr>
<td>IndoChina</td>
<td>2.02</td>
<td>2.7</td>
<td>2.7</td>
<td>IndoChina</td>
<td>2.35</td>
<td>2.7</td>
</tr>
</tbody>
</table>

Geography does not seem to help for Flickr graph.
FLICKR: COMPRESSIBILITY OVER TIME
A PROPERTY OF SHINGLE ORDERING

*Theorem*. Using shingle ordering, a constant fraction of edges will be “copied” in graphs generated by preferential attachment/copying models.

- Preferential attachment model: Rich get richer – a new node links to an existing node with probability proportional to its degree.
- Shows that shingle ordering helps BV/BL-style compressions in stylized graph models.
GAP DISTRIBUTION

Shingle ordering produces smaller gaps
WHO IS THE CULPRIT

Low degree nodes are responsible for incompressibility
COMPRESSION-FRIENDLY ORDERINGS

- In BV/BL, canonical order is all that matters
- Problem. Given a graph, find the canonical ordering that will produce the best compression in BV/BL
  - The ordering should capture locality and similarity
  - The ordering must help BV/BL-style compressions
- We propose two formulations of this problem
MLogA formulation

MLogA. Find an ordering \( p \) of nodes such that

\[
\sum_{(u, v) \in E} \lg |\pi(u) - \pi(v)|
\]

is minimized

- Minimize sum of encoding gaps of edges
- Without \( \lg \), this is min linear arrangement (MLinA)
- MLinA is well-studied ((\( \log n \) \( \log \log n \)) approximable, ...
- MLinA and MLogA are very different problems

**Theorem.** MLogA is NP-hard

- Proof using the inapproximability of MaxCut
**MLogGapA formulation**

- MLogGapA. For an ordering \( p \), let \( f_\pi(u) = \text{cost of compressing the out-neighbors of } u \text{ under } \pi \)
- If \( u_1, \ldots, u_k \) are out-neighbors ordered wrt \( \pi \), \( u_0 = u \)
  \[
  f_\pi(u) = \sum_{i=1}^{k} |g| |\pi(u_i) - \pi(u_{i-1})|
  \]
- Find an ordering \( \pi \) of nodes to minimize \( \sum_u f_\pi(u) \)
- Minimize encoding gaps of neighbors of a node
- MLogGapA and MLogA are very different problems
  - *Theorem*. MLinGapA is NP-hard
  - *Conjecture*. MLogGapA is NP-hard
**Summary**

- Social networks appear to be not very compressible
- Host graphs are equally challenging

- These two graphs are very unlike the web graph, which is highly compressible

**Future directions**

- Can we compress social networks better? *Boldi, Santini, Vigna 2009*
- Is there a lower bound on incompressibility? Our analysis applies only to BV-style compressions
- Algorithmic questions: Hardness of MLogGapA, Good approximation algorithms
- Modeling: Compressibility of existing graph models, More nuanced models for the compressible web *Chierichetti, Kumar, Lattanzi, Mitzenmacher, Panconesi, Raghavan FOCS 2009*
REFERENCES

THE END

- Thank You