Compressing Social Networks
The Minimum Logarithmic Arrangement Problem

Chad Waters

School of Computing
Clemson University
cgwater@clemson.edu

March 4, 2013
Determine the extent to which social networks can be compressed.

- Like Web graph, typical queries seek the neighbors of a node.
- Need a model that facilitates efficient adjacency queries.
- Social networks are not random graphs.
- Exhibit distinctive local properties, such as degree sequences.
- Compressibility $\approx$ “Randomness”
Determine the extent to which social networks can be compressed.

- Like Web graph, typical queries seek the neighbors of a node.
- Need a model that facilitates efficient adjacency queries.
- Social networks are not random graphs.
- Exhibit distinctive local properties, such as degree sequences.
- Compressibility $\approx$ “Randomness”
Web Graphs

Web graphs are a special case of a social network.

- Known to be highly compressible.
- Exploit lexicographic locality: order URLs naturally.
- Proximal pages have similar neighborhoods.

Lexicographic ordering produces the following properties:

- *Similarity*: Proximal pages have similar sets of neighbors.
- *Locality*: Many links are in same domain, and share neighbors.

No obvious natural ordering for general social networks.
Web graphs are a special case of a social network.

- Known to be highly compressible.
- Exploit lexicographic locality: order URLs naturally.
- Proximal pages have similar neighborhoods.

Lexicographic ordering produces the following properties:

- *Similarity*: Proximal pages have similar sets of neighbors.
- *Locality*: Many links are in the same domain, and share neighbors.

No obvious natural ordering for general social networks.
Web graphs are a special case of a social network.

- Known to be highly compressible.
- Exploit lexicographic locality: order URLs naturally.
- Proximal pages have similar neighborhoods.

Lexicographic ordering produces the following properties:

- **Similarity**: Proximal pages have similar sets of neighbors.
- **Locality**: Many links are in same domain, and share neighbors.

No obvious natural ordering for general social networks.
Need to leverage a different feature..

- Lexicographic ordering not reasonable for true social networks.
- Need an ordering that rewards link reciprocity.
Boldi-Vigna Compression Scheme

Incorporates three main ideas

1. If many nodes have similar neighborhoods, express a neighborhood in terms of a node with a similar neighborhood.
2. If destinations of edges exhibit locality, small integers can be used to encode them.
3. Use *gap encodings* to store a sequence of edge destinations.
Boldi-Vigna Compression Scheme

Incorporates three main ideas

1. If many nodes have similar neighborhoods, express a neighborhood in terms of a node with a similar neighborhood.
2. If destinations of edges exhibit locality, small integers can be used to encode them.
3. Use *gap encodings* to store a sequence of edge destinations.
Boldi-Vigna Compression Scheme

Incorporates three main ideas

1. If many nodes have similar neighborhoods, express a neighborhood in terms of a node with a similar neighborhood.
2. If destinations of edges exhibit locality, small integers can be used to encode them.
3. Use *gap encodings* to store a sequence of edge destinations.
Boldi-Vigna Compression Scheme

BV scheme for compressing Web graphs.

- Nodes are Web pages, edges are hyperlinks between them.
- Web pages are ordered lexicographically by URL.
- Let $w$ be a window parameter (BV recommends $w = 8$)
Boldi-Vigna Compression Scheme

Consider a web page $v$. First a *copy* phase:

- Examine the list $\text{out}(v)$ of $v$’s out neighbors.
- Check if it is similar to one of the $w - 1$ preceding Web pages.
- Call this prototype page $u$, if it exists.

Followed by an *encoding* phase:

- If we have a prototype, encode the offset with $\log w$ bits.
- Then write the changes between $\text{out}(v)$ and $\text{out}(u)$.
- Otherwise, write $\log w$ zeroes, followed by the list $\text{out}(v)$.

Benefits from a natural ordering with locality on the Web pages.
Boldi-Vigna Compression Scheme

Consider a web page \( v \). First a *copy* phase:

- Examine the list \( \text{out}(v) \) of \( v \)’s out neighbors.
- Check if it is similar to one of the \( w - 1 \) preceding Web pages.
- Call this prototype page \( u \), if it exists.

Followed by an *encoding* phase:

- If we have a prototype, encode the offset with \( \log_w \) bits.
- Then write the changes between \( \text{out}(v) \) and \( \text{out}(u) \).
- Otherwise, write \( \log_w \) zeroes, followed by the list \( \text{out}(v) \).

Benefits from a natural ordering with locality on the Web pages.
Boldi-Vigna Compression Scheme

Consider a web page \( v \). First a *copy* phase:

- Examine the list \( \text{out}(v) \) of \( v \)'s out neighbors.
- Check if it is similar to one of the \( w - 1 \) preceding Web pages.
- Call this prototype page \( u \), if it exists.

Followed by an *encoding* phase:

- If we have a prototype, encode the offset with \( \log w \) bits.
- Then write the changes between \( \text{out}(v) \) and \( \text{out}(u) \).
- Otherwise, write \( \log w \) zeroes, followed by the list \( \text{out}(v) \).

Benefits from a natural ordering with locality on the Web pages.
This scheme has two nice properties:

1. Dependent only on locality given by a canonical ordering.
2. Adjacency queries can be served extremely quickly.

Decode neighbors backwards through prototype chain.
A modification to the BV compression scheme leveraging observed properties of social networks, specifically *link reciprocity*. 
Consider a node \( v \). First a *copy* phase:

- Again, encode \( v \) as changes between it and a prototype \( u \).
- If no prototype can be found, no copying will occur.
- A bit added for each of \( \text{out}(u) \) signifying if it is in \( \text{out}(v) \).

Then we record *residual edges*:

- Let \( v_1 \leq \ldots \leq v_k \) be the out-neighbors of \( v \).
- Encode the gaps: \( |v_1 - v|, |v_2 - v_1|, \ldots, |v_k - v_{k-1}| \)

And remove the redundancy of *reciprocal edges*:

- Encode the reciprocal out-neighbors of \( v \).
- For each \( v' \in \text{out}(v) \) such that \( v' > v \), discard \((v', v)\) if \( v \in \text{out}(v') \) (the edge is reciprocal).
Consider a node \( v \). First a *copy* phase:

- Again, encode \( v \) as changes between it and a prototype \( u \).
- If no prototype can be found, no copying will occur.
- A bit added for each of \( \text{out}(u) \) signifying if it is in \( \text{out}(v) \).

Then we record *residual edges*:

- Let \( v_1 \leq \ldots \leq v_k \) be the out-neighbors of \( v \).
- Encode the gaps: \( |v_1 - v|, |v_2 - v_1|, \ldots, |v_k - v_{k-1}| \)

And remove the redundancy of *reciprocal edges*:

- Encode the reciprocal out-neighbors of \( v \).
- For each \( v' \in \text{out}(v) \) such that \( v' > v \), discard \((v', v)\) if \( v \in \text{out}(v') \) (the edge is reciprocal).
Consider a node \( v \). First a *copy* phase:

- Again, encode \( v \) as changes between it and a prototype \( u \).
- If no prototype can be found, no copying will occur.
- A bit added for each of \( \text{out}(u) \) signifying if it is in \( \text{out}(v) \).

Then we record *residual edges*:

- Let \( v_1 \leq \ldots \leq v_k \) be the out-neighbors of \( v \).
- Encode the gaps: \( |v_1 - v|, |v_2 - v_1|, \ldots, |v_k - v_{k-1}| \)

And remove the redundancy of *reciprocal edges*:

- Encode the reciprocal out-neighbors of \( v \).
- For each \( v' \in \text{out}(v) \) such that \( v' > v \), discard \((v', v)\) if \( v \in \text{out}(v') \) (the edge is reciprocal).
This method potentially outperforms BV in terms of compression.

- Redundant edges are compressed out.

Adjacency queries may be slower than with BV:

- BL does not bound length of prototype chains.
- If performance degrades, a limit can be implemented.
This method potentially outperforms BV in terms of compression.

- Redundant edges are compressed out.

Adjacency queries may be slower than with BV:

- BL does not bound length of prototype chains.
- If performance degrades, a limit can be implemented.
Performance of both the Boldi-Vigna and Backlinks compression schemes depends heavily on the ordering of nodes. We wish to find an ordering that is practical for graphs that may not admit an obvious natural ordering.

We wish to find an ordering that provides both locality and similarity, just as lexicographic ordering gave us with the Web graphs. This leads us to the minimum logarithmic arrangement problem.
**MinLogA Problem**

Find a permutation $\pi : V \rightarrow [n]$ such that

$$\sum_{(u,v) \in E} \log |\pi(u) - \pi(v)|$$

is minimized.

Using this cost $\log |\pi(u) - \pi(v)|$ represents the information-theoretic optimal ordering.
Repeating the cost in this equation with simply $|\pi(u) - \pi(v)|$, we get the *minimum linear arrangement* problem.

- Known to be NP-Hard.
- Not much is known about its approximability.
- Best known algorithm approximates to $O(\sqrt{\log n \log \log n})$
- Ambühl et al rule out the existence of a PTAS.

Fundamentally different problem than **MinLogA** in that a solution to one is not necessarily a solution to the other in the general case.
**MinLogGapA Problem**

It is always more efficient to compress the gaps induced by the neighbors of a node.

- Suppose \( u < v_1 < v_2 \) and \((u, v_1), (u, v_2) \in E\).
- Compressing \( v_1 - u \) and \( v_2 - v_1 \) is always less expensive.

Therefore, the problem we really want to solve is:

**MinLogGapA**

Find a permutation \( \pi : V \rightarrow [n] \) such that

\[
\sum_{u \in V} \sum_{i=1}^{k} \log |\pi(u_i) - \pi(u_{i-1})|
\]

is minimized, where \( u_0 = u \) and \( \{u_1, \ldots, u_k\} = \text{out}(u) \).
MinLogGapA Problem

It is always more efficient to compress the gaps induced by the neighbors of a node.

- Suppose $u < v_1 < v_2$ and $(u, v_1), (u, v_2) \in E$.
- Compressing $v_1 - u$ and $v_2 - v_1$ is always less expensive.

Therefore, the problem we really want to solve is:

MinLogGapA

Find a permutation $\pi : V \rightarrow [n]$ such that

$$\sum_{u \in V} \sum_{i=1}^{k} \log |\pi(u) - \pi(u_{i-1})|$$

is minimized, where $u_0 = u$ and $\{u_1, \ldots, u_k\} = \text{out}(u)$. 
Other Orderings

Lindstrom suggests a few more ordering metrics:

\( p \)-sum (equivalent to \text{MINLINA} where \( p = 1 \)):

\[
\min \sigma_p = \lim_{q \to p} \left( \sum_{uv \in E} |\pi(u) - \pi(v)|^q \right)^{1/q}
\]

\( p \)-mean:

\[
\min \mu_p = \lim_{q \to p} \left( \frac{1}{|E|} \sum_{uv \in E} |\pi(u) - \pi(v)|^q \right)^{1/q}
\]

Which, by way of the geometric mean, is the \textit{minimum edge product problem} for \( \mu_0 \):

\[
\mu_0 = \exp \left( \frac{1}{|E|} \sum_{uv \in E} \log |\pi(u) - \pi(v)| \right) = \left( \prod_{uv \in E} |\pi(u) - \pi(v)| \right)^{1/|E|}
\]
We wish to find a practical heuristic for both MinLogA and MinLogGapA. Based on concept of obtaining a “fingerprint” for the out-neighbors for each node, and sorting on this fingerprint.

**Jaccard Coefficient**

*A notion of similarity between two sets, defined as*

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$
Let $\sigma$ be a random permutation of the elements in $A \cup B$

and let $M_\sigma(A) = \sigma^{-1}(\min_{a \in A} \sigma(a))$, the first element in $A_\sigma$. 
Let $\sigma$ be a random permutation of the elements in $A \cup B$ and let $M_\sigma(A) = \sigma^{-1}(\min_{a \in A} \sigma(a))$, the first element in $A_\sigma$.

$$\Pr[M_\sigma(A) = M_\sigma(B)] =$$
Let $\sigma$ be a random permutation of the elements in $A \cup B$

and let $M_\sigma(A) = \sigma^{-1}(\min_{a \in A} \sigma(a))$, the first element in $A_\sigma$.

$$\Pr[M_\sigma(A) = M_\sigma(B)] = \frac{|A \cap B|}{|A \cup B|} = J(A, B)$$
Shingle Ordering Heuristic

Treat the out-neighbors of each $u \in V$ as a set.

- Compute the shingle $M_\sigma(\text{out}(u))$ for a suitable $\sigma$.
- Nodes in $V$ can now be ordered by their shingles.
- If two nodes share a large number of out-neighbors, with high probability, they will have the same shingle and will be near each other in the shingle ordering.
- Therefore, shingle ordering maintains the properties of locality and similarity.
Shingle ordering and double-shingle ordering performs better than almost all baseline sorting methods:

- **Random order**
- **Natural order** - Most obvious ordering. For Web and host graphs, these would be lexicographic order on URL. For social networks, crawl ordering.
- **DFS/BFS order** - Performs almost as poorly as random ordering
- **Gray code order** - \( v \) precedes \( w \) if the row of \( v \) precedes the row of \( w \) in the binary adjacency matrix, according to the canonical Gray code
- **Geographic order** - Liben-Nowell et al, show that \( \sim 70\% \) of social network edges “arise from geographical proximity”.
Compression Performance

Shingle ordering and double-shingle ordering performs better than almost all baseline sorting methods:

- **Random order**

- **Natural order** - Most obvious ordering. For Web and host graphs, these would be lexicographic order on URL. For social networks, crawl ordering.

- **DFS/BFS order** - Performs almost as poorly as random ordering

- **Gray code order** - $v$ precedes $w$ if the row of $v$ precedes the row of $w$ in the binary adjacency matrix, according to the canonical Gray code

- **Geographic order** - Liben-Nowell et al, show that $\sim 70\%$ of social network edges “arise from geographical proximity”.
Shingle ordering and double-shingle ordering performs better than almost all baseline sorting methods:

- **Random order**
- **Natural order** - Most obvious ordering. For Web and host graphs, these would be lexicographic order on URL. For social networks, crawl ordering.
- **DFS/BFS order** - Performs almost as poorly as random ordering
- **Gray code order** - $v$ precedes $w$ if the row of $v$ precedes the row of $w$ in the binary adjacency matrix, according to the canonical Gray code
- **Geographic order** - Liben-Nowell et al, show that $\sim 70\%$ of social network edges “arise from geographical proximity”.
Shingle ordering and double-shingle ordering performs better than almost all baseline sorting methods:

- **Random order**
- **Natural order** - Most obvious ordering. For Web and host graphs, these would be lexicographic order on URL. For social networks, crawl ordering.
- **DFS/BFS order** - Performs almost as poorly as random ordering
- **Gray code order** - \( v \) precedes \( w \) if the row of \( v \) precedes the row of \( w \) in the binary adjacency matrix, according to the canonical Gray code
- **Geographic order** - Liben-Nowell et al, show that \( \sim 70\% \) of social network edges “arise from geographical proximity”.
Shingle ordering and double-shingle ordering performs better than almost all baseline sorting methods:

- **Random order**
- **Natural order** - Most obvious ordering. For Web and host graphs, these would be lexicographic order on URL. For social networks, crawl ordering.
- **DFS/BFS order** - Performs almost as poorly as random ordering
- **Gray code order** - $v$ precedes $w$ if the row of $v$ precedes the row of $w$ in the binary adjacency matrix, according to the canonical Gray code
- **Geographic order** - Liben-Nowell et al, show that $\sim 70\%$ of social network edges “arise from geographical proximity”.
Why does Shingle Ordering Work?

For most graphs, we increase the number of small gaps, which can be encoded succinctly. The notable exception is with the LiveJournal graph, where the crawl ordering performed marginally better.

Minimizing the size of the gaps and increasing the number of small gaps (within some threshold window) equates to a performance boost for both BV and BL.
Why might a graph be hard to compress?

Which compresses better: dense or sparse portions of a graph?
Why might a graph be hard to compress?

Which compresses better: dense or sparse portions of a graph?

The low-degree nodes in social networks are to blame.

We can compress $k$-cores separately (down to virtual nodes).
The analogous structure of Web graphs and social networks is unmistakable. Social networks do not, however, admit a natural ordering that consistently performs as well as lexicographic ordering does for Web graphs. Shingle ordering does beat most other baseline orderings, and highlights the value of link reciprocity.