Pushing the Envelope in Graph Compression

Panagiotis Liakos\textsuperscript{1}  \hspace{0.5cm} Katia Papakonstantinou\textsuperscript{1} \hspace{0.5cm} Michael Sioutis\textsuperscript{2}

\textsuperscript{1}University of Athens, \textsuperscript{2}Université Lille-Nord de France, CRIL-CNRS

23\textsuperscript{rd} ACM International Conference on Information and Knowledge Management
Shanghai, November 6th, 2014
Motivation

- Explosive growth of large-scale systems modeled as graphs
  - Web Graphs
  - Social Networks
- Critical applications (e.g., in the field of Graph Mining) need in-memory graph representations
  - Serving adjacency queries
  - Maintaining snapshots for archival purposes

We need efficient compressed graph representations!
Graph compression

Graph compression / Compact graph representation:

Find a compressed graph representation that allows mining without decompressing the graph.

We are interested in graphs created by human activity (e.g., web graphs, social network graphs). They exhibit, under certain orderings, common properties:

- power law distributed degrees,
- locality (of reference) and
- copy (or similarity) property

which induce redundancy in the graphs’ representations, and are taken into account in the design of compression methods.
Graph compression methods

Algorithms for compressing web graphs
- adjacency lists
- combination of adjacency lists and matrices

Algorithms for compressing social network graphs
- adjacency matrices

Reordering algorithms that lead to higher compression

State of the art is BV: Further minimize space / access time
1 State of the art

2 Our method

3 Our results
Laboratory for Web Algorithmics’ (LAW) techniques

- **Modified gap representation [BV04]**
  - Represent each list of successors as a list of gaps to exploit the *locality of reference*

- **Reference list [BV04]**
  - Code each adjacency list as a “modified” version of a previous list to exploit *similarity (copy-property)*

- **Layered Label Propagation [BRSV11]**
  - Reorders very large graphs to provide a major increase in compression with respect to prior efforts
Layered Label Propagation effect

Graph cit-Patents before and after LLP reordering:

Observation: The area around the main diagonal is dense under certain orderings [LPS14a]
Main idea

As the area around the main diagonal of the graph’s adjacency matrix is dense after reordering, isolate it and compress it separately.

Heat maps of the adjacency matrices of:

(i) cnr-2000
(ii) web-Stanford
(iii) roadNet-CA
(iv) roadNet-PA
(v) dblp-2010
(vi) cit-Patents
(vii) amazon-2008
(viii) ljournal-2008
(ix) twitter-2010
Our method in a glance

Isolate a dense subgraph $S$ around the main diagonal

Apply data compression on $S$

Compress $G \setminus S$ using the state-of-the-art method of Boldi et al. [BV04, BRSV11] (or any other efficient graph compression method)

**Algorithm 1: BV+(G, $k$, $b$)**

```
input : A directed graph $G = (V, E)$ and parameters $k$ and $b$.
output : A compressed representation of $G$.
begin

setNonD ← set();
$k$-diagonalStripe ← array(array([000 . . 0]) × |V|); 2k+1 bits

foreach $(u, v) \in E$ do
  if $u - k \leq v \leq u + k$ then
    $k$-diagonalStripe[$u$][$v$] ← 1;
  else
    setNonD ← setNonD ∪ $(u, v)$;

seqDict ← dict();
foreach seq ∈ $k$-diagonalStripe do
  if seq ∉ seqDict then
    seqDict[seq] ← 1;
  else
    seqDict[seq]++;

foreach (key, value) ∈ seqDict do
  seqDict[key] ← value × # of 1s ∈ key;
seqDict ← sort seqDict by value (desc. order);
seqSet ← {first $2^b - 1$ sequences (keys) of seqDict};
foreach seq ∈ $k$-diagonalStripe do
  if seq ∈ seqSet then
    use $b$ bits to compress seq;
  else
    bestSeq ← bestSubset(seqSet, seq, $k$);
    use $b$ bits to compress bestSeq;
    setNonD ← setNonD ∪ {edges of seq that were left out of bestSeq};

compress setNonD using BV;
```
Our techniques

- Isolating a dense subgraph
- Data compression
Diagonal stripe: For $G = (V, E)$ and $k \in \mathbb{Z}_+$, the $k$-diagonal stripe of $G$ is the set: $\{(i, j) \mid i - k \leq j \leq i + k \text{ and } i, j \in \{0, \ldots, |V|\}\}$.
Isolating a dense subgraph

**Diagonal stripe:** For $G = (V, E)$ and $k \in \mathbb{Z}_+$, the $k$-diagonal stripe of $G$ is the set: $\{(i, j) \mid i - k \leq j \leq i + k$ and $i, j \in \{0, \ldots, |V|\}\}$. 

The bits/edge required to store the stripe increases with $k$. However, even a sparser stripe may lead to higher compression of the graph!
The computation of the bits/edge needed to represent the diagonal stripe of a given graph is straightforward:

\[
\text{uncompressed diagonal ratio} = \frac{(2k+1)|V|}{|E|} = \frac{15|V|}{0.6841|E|} = 7.76 \text{ bits/edge.}
\]

roadNet-PA has 68.41\% of its edges in the 7-diagonal

However, \(BV\) yields a compression ratio of 12.86 bits/edge for roadNet-PA.

Can we do better?
Based on Shannon’s source coding theorem, we get an indication of the expected compression ratio for the diagonal stripe.

**Proposition 1.**

Consider $k \in \mathbb{Z}_+$ and a graph $G = (V, E)$ with a percentage $p$ of its edges belonging in the $k$-diagonal stripe. The minimum expected compression ratio of the diagonal stripe is upper bounded by

$$\frac{\log \left( \frac{(2k+1)|V|}{p|E|} \right)}{p|E|} \text{ bits/edge}.$$  

**Estimation of the compressed roadNet-PA diagonal ratio**

Expected compression ratio $= \frac{\log \left( \frac{15|V|}{0.6841|E|} \right)}{0.6841|E|} = 2.98 \text{ bits/edge} << 12.86 (BV)$

Our method is likely to offer a significant improvement!
Data compression on the dense subgraph

Consider the following graph and assume $k = 3$ and $b = 2$.

- $\frac{36}{47}$ edges are held in $\frac{b}{2k+1} = \frac{2}{7}$ of the original diagonal space!
- The “lost” edges are passed on to the next step ($\sim$ lossless)

**roadNet-PA compressed diagonal ratio**

$k = 7$, $b = 2 \Rightarrow 1.95$ bits/edge.
Putting everything together

- **Size of compressed graph:** \( b|V| + S_{BV} \)

<table>
<thead>
<tr>
<th>Time Complexity</th>
<th>Edges in comp. diagonal</th>
<th>Edges outside comp. diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge Exists</strong></td>
<td>( O(1) )</td>
<td>( &lt; BV )</td>
</tr>
<tr>
<td><strong>Successors</strong></td>
<td>( O(b) )</td>
<td>( &lt; BV )</td>
</tr>
</tbody>
</table>

Katia Papakonstantinopoulou (University of Athens)
## Experimental evaluation: Compression ratios

<table>
<thead>
<tr>
<th>graph</th>
<th># nodes</th>
<th># edges</th>
<th>% of edges in diagonal</th>
<th>compression ratio (bits/edge)</th>
<th>BV+ parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>cnr-2000</td>
<td>325,557</td>
<td>3,216,152</td>
<td>6.05 %</td>
<td>3.71 BV+ 3.62</td>
<td>17 2 ▼</td>
</tr>
<tr>
<td>web-Stanford</td>
<td>281,903</td>
<td>3,985,272</td>
<td>6.78 %</td>
<td>4.06 BV+ 3.90</td>
<td>1 2 ▼</td>
</tr>
<tr>
<td>roadnet-CA</td>
<td>1,965,206</td>
<td>5,533,214</td>
<td>64.50 %</td>
<td>13.30 BV+ 10.58</td>
<td>7 6 ▼</td>
</tr>
<tr>
<td>roadnet-PA</td>
<td>1,088,092</td>
<td>3,083,796</td>
<td>66.89 %</td>
<td>12.86 BV+ 10.07</td>
<td>7 6 ▼</td>
</tr>
<tr>
<td>dblp-2010</td>
<td>326,186</td>
<td>1,615,400</td>
<td>54.49 %</td>
<td>8.63 BV+ 7.2</td>
<td>24 7 ▼</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3,774,767</td>
<td>33,037,894</td>
<td>19.07 %</td>
<td>14.72 BV+ 14.25</td>
<td>9 6 ▼</td>
</tr>
<tr>
<td>amazon-2008</td>
<td>735,323</td>
<td>5,158,388</td>
<td>59.27 %</td>
<td>10.77 BV+ 10.07</td>
<td>23 15 ▼</td>
</tr>
<tr>
<td>ljournal-2008</td>
<td>5,363,260</td>
<td>79,023,142</td>
<td>7.65 %</td>
<td>11.84 BV+ 11.78</td>
<td>2 4 ▼</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>41,652,230</td>
<td>1,468,365,182</td>
<td>2.58 %</td>
<td>14.52 BV+ 14.42</td>
<td>17 6 ▼</td>
</tr>
</tbody>
</table>

- Large impact on social network graphs (dblp-2010: 16.6%)
- Impressive results for road network graphs (> 20%)
- Improvements on web graphs (2.4% – 3.9%)
Experimental evaluation: Access times (ms)

<table>
<thead>
<tr>
<th></th>
<th>cnr-2000</th>
<th>roadNet-CA</th>
<th>ljournal-2008</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge Exists</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BV+ (D)</td>
<td>645</td>
<td>600</td>
<td>663</td>
</tr>
<tr>
<td>BV+ (Non-D)</td>
<td>2,286</td>
<td>832</td>
<td>4,373</td>
</tr>
<tr>
<td>BV</td>
<td>2,397</td>
<td>923</td>
<td>4,518</td>
</tr>
<tr>
<td><strong>Successors</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BV+ (D)</td>
<td>643</td>
<td>668</td>
<td>627</td>
</tr>
<tr>
<td>BV+ (Non-D)</td>
<td>1,947</td>
<td>849</td>
<td>1,940</td>
</tr>
<tr>
<td>BV</td>
<td>2,159</td>
<td>891</td>
<td>2,009</td>
</tr>
</tbody>
</table>

- Significantly faster for the compressed diagonal part
- Faster than $BV$ for the rest of the graph

We outscore $BV$ in any non-single core environment
Effect of parameters

We can estimate a good $k$ using Proposition 1.

Delicate balance between:

- Minimizing the diagonal stripe ratio
- Easing the task of compressing the rest of the graph

For our dataset: $k \in [2, 20]$ and $b \leq k$

**Best choice for roadNet-PA**

$k = 7, b = 6 \Rightarrow 3.27$ bits/edge (although it is $> 1.95$ bits/edge).


We go beyond the state-of-the-art compressed data structure of Boldi et al. [BV04, BRSV11] for web and social graphs by further exploiting the clustering properties (locality, similarity) observed in these graphs.

Our implementation can easily be employed to improve any compression method.

We reduced the best recorded compression ratios of well-known datasets up to 21.7% and the corresponding access times up to 21.13%.

We are currently examining labelings that favor our compression method.
Thank you!

For further details refer to:
http://hive.di.uoa.gr/network-analysis

or email me at: katia@di.uoa.gr
Modified gap representation

Represent each list of successors as a list of gaps to exploit the *locality of reference*.

<table>
<thead>
<tr>
<th>Node</th>
<th>Outdegree</th>
<th>Successors</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>15, 16, 17, 22, 23, 24, 315, 316, 317, 3041</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>13, 15, 16, 17, 50</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Outdegree</th>
<th>Successors</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>3, 1, 0, 0, 0, 0, 3, 0, 178, 111, 718</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>1, 0, 0, 4, 0, 0, 290, 0, 0, 2723</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>9, 1, 0, 0, 32</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Code each adjacency list as a “modified” version of a previous list to exploit similarity (copy-property)

<table>
<thead>
<tr>
<th>Node</th>
<th>Outdegree</th>
<th>Successors</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>15, 16, 17, 22, 23, 24, 315, 316, 317, 3041</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>13, 15, 16, 17, 50</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Outdegree</th>
<th>Reference</th>
<th>Copy list</th>
<th>Extra nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>0</td>
<td>01110011010</td>
<td>13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>1</td>
<td>01110011010</td>
<td>22, 316, 317, 3041</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>3</td>
<td>11110000000</td>
<td>50</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>