In-core computation of distance distributions and geometric centralities with HyperBall: A hundred billion nodes and beyond

Paolo Boldi, Sebastiano Vigna
Laboratory for Web Algorithmics
Università degli Studi di Milano, Italy
Setup
Setup

- You have a very large graph (social, web)
Setup

- You have a very large graph (social, web)
- You want to understand something of its *global* structure (not triangles/degree distribution/etc.)
Setup

- You have a very large graph (social, web)
- You want to understand something of its global structure (not triangles/degree distribution/etc.)
- First candidate: distance distribution (and, in the directed case, the number of reachable pairs)
Setup

✦ You have a very large graph (social, web)
✦ You want to understand something of its *global* structure (not triangles/degree distribution/etc.)
✦ First candidate: *distance distribution* (and, in the directed case, the number of *reachable pairs*)
✦ You want to understand which nodes are *important* in some sense
For real
For real

- First paper at WWW 2011 (with Marco Rosa)
For real

- First paper at WWW 2011 (with Marco Rosa)
- Open-source software part of the WebGraph framework
For real

✦ First paper at WWW 2011 (with Marco Rosa)
✦ Open-source software part of the WebGraph framework
✦ Run on Facebook (whole graph) using just
For real

✦ First paper at WWW 2011 (with Marco Rosa)
✦ Open-source software part of the WebGraph framework
✦ Run on Facebook (whole graph) using just
Geometric Centralities
Geometric Centralities

- Closeness (Bavelas 1946):
Geometric Centralities

- Closeness (Bavelas 1946): \( \frac{1}{\sum_y d(y, x)} \)
Geometric Centralities

- Closeness (Bavelas 1946): \[ \frac{1}{\sum_y d(y, x)} \]
- The summation is over all \( y \) such that \( d(y, x) < \infty \)
Geometric Centralities

- Closeness (Bavelas 1946):
  \[ \frac{1}{\sum_y d(y, x)} \]

- The summation is over all \( y \) such that \( d(y, x) < \infty \)

- Harmonic centrality:
Geometric Centralities

- Closeness (Bavelas 1946): \[ \frac{1}{\sum_y d(y, x)} \]
- The summation is over all \( y \) such that \( d(y, x) < \infty \)
- Harmonic centrality: \[ \sum_{y \neq x} \frac{1}{d(y, x)} \]
Why?
Why?

- Using HyperBall, we were able to evaluate geometric centrality in an IR setting
Why?

- Using HyperBall, we were able to evaluate geometric centrality in an IR setting.
- The (preliminary) results show that harmonic centrality has a very good signal (in fact, better NDCG@10/P@10 than anything we tried).
Why?

- Using HyperBall, we were able to evaluate geometric centrality in an IR setting.
- The (preliminary) results show that harmonic centrality has a very good signal (in fact, better NDCG@10/P@10 than anything we tried).
- In general, HyperBall makes it possible to use harmonic centrality on very large graphs.
Hollywood: PageRank

Ron Jeremy
Adolf Hitler
Lloyd Kaufman
George W. Bush

Ronald Reagan
Bill Clinton
Martin Sheen
Debbie Rochon
Hollywood: Harmonic

George Clooney  Samuel Jackson  Sharon Stone  Tom Hanks

Martin Sheen  Dennis Hopper  Antonio Banderas  Madonna
Intermediate step
Intermediate step

- For each node, we compute in sequence the number of nodes at distance exactly $t$
Intermediate step

- For each node, we compute in sequence the number of nodes at distance exactly $t$
- Adding up over all nodes, we get the distance distribution (modulo normalization)
Intermediate step

- For each node, we compute in sequence the number of nodes at distance exactly $t$
- Adding up over all nodes, we get the distance distribution (modulo normalization)
- Centralities can be rewritten, e.g., harmonic:
Intermediate step

- For each node, we compute in sequence the number of nodes at distance exactly $t$
- Adding up over all nodes, we get the distance distribution (modulo normalization)
- Centralities can be rewritten, e.g., harmonic:

$$\sum_{t>0} \frac{1}{t} |\{y \mid d(y, x) = t\}|$$
How do you compute it?
How do you compute it?

- Many many breadth-first visits: $O(mn)$, needs direct access
How do you compute it?

- Many many breadth-first visits: $O(mn)$, needs direct access
- Sampling: a fraction of breadth-first visits, very unreliable results on graphs that are not strongly connected, needs direct access
How do you compute it?

- Many many breadth-first visits: \(O(mn)\), needs direct access
- Sampling: a fraction of breadth-first visits, very unreliable results on graphs that are not strongly connected, needs direct access
- Edith Cohen’s [JCSS 1997] size estimation framework: very powerful but does not scale or parallelize really well, needs direct access
Alternative: Diffusion
Alternative: Diffusion

- Basic idea: Palmer et. al, KDD '02
Alternative: Diffusion

- Basic idea: Palmer et. al, KDD ’02
- Let $B_t(x)$ be the ball of radius $t$ around $x$ (nodes at distance at most $t$ from $x$)
Alternative: Diffusion

- Basic idea: Palmer et. al, KDD ’02
- Let $B_t(x)$ be the ball of radius $t$ around $x$ (nodes at distance at most $t$ from $x$)
- Clearly $B_0(x) = \{x\}$
Alternative: Diffusion

- Basic idea: Palmer et. al, KDD ’02
- Let $B_t(x)$ be the ball of radius $t$ around $x$ (nodes at distance at most $t$ from $x$)
- Clearly $B_0(x)={}x$
- But also $B_{t+1}(x)=\bigcup_{x\rightarrow y}B_t(y)\cup\{x\}$
Alternative: Diffusion

- Basic idea: Palmer et al, KDD ’02
- Let $B_t(x)$ be the ball of radius $t$ around $x$ (nodes at distance at most $t$ from $x$)
- Clearly $B_0(x) = \{x\}$
- But also $B_{t+1}(x) = \bigcup_{x \rightarrow y} B_t(y) \bigcup \{x\}$
- So we can compute balls by enumerating the arcs $x \rightarrow y$ and performing set unions
A round of updates

[Diagram of a series of updates]
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
A round of updates
Another round...
Another round...
Another round...
Another round...
Another round...
Another round...
Another round...
Another round...
Easy but expensive
Easy but expensive

- Each set uses linear space; overall quadratic
Easy but expensive

✦ Each set uses linear space; overall quadratic
✦ Impossible!
Easy but expensive

- Each set uses linear space; overall quadratic
- Impossible!
- But what if we use approximate sets?
Easy but expensive

✦ Each set uses linear space; overall quadratic
✦ Impossible!
✦ But what if we use approximate sets?
✦ Idea: use probabilistic counters, which represent sets but answer just to “size?” questions
Easy but expensive

- Each set uses linear space; overall quadratic
- Impossible!
- But what if we use approximate sets?
- Idea: use probabilistic counters, which represent sets but answer just to “size?” questions
- Very small!
Main trick
Main trick

- Choose an approximate set such that unions can be computed quickly
Main trick

- Choose an approximate set such that unions can be computed quickly
- ANF [Palmer et al., KDD ’02] uses Martin–Flajolet (MF) counters ($\log n + c$ space)
Main trick

- Choose an approximate set such that unions can be computed quickly
- ANF [Palmer et al., KDD ’02] uses Martin–Flajolet (MF) counters (log \( n + c \) space)
- We use HyperLogLog counters [Flajolet et al., 2007] (log log \( n \) space)
Main trick

- Choose an approximate set such that unions can be computed quickly
- ANF [Palmer et al., KDD ’02] uses Martin–Flajolet (MF) counters (log $n + c$ space)
- We use HyperLogLog counters [Flajolet et al., 2007] (log log $n$ space)
- MF counters can be combined with an OR
Main trick

- Choose an approximate set such that unions can be computed quickly
- ANF [Palmer et al., KDD ’02] uses Martin–Flajolet (MF) counters (\(\log n + c\) space)
- We use HyperLogLog counters [Flajolet et al., 2007] (\(\log \log n\) space)
- MF counters can be combined with an OR
- We use *broadword programming* to combine HyperLogLog counters quickly!
HyperLogLog counters
HyperLogLog counters

- Instead of actually counting, we observe a statistical feature of a set (think stream) of elements
HyperLogLog counters

- Instead of actually counting, we observe a statistical feature of a set (think stream) of elements
- The feature: the number of trailing zeroes of the value of a very good hash function
HyperLogLog counters

- Instead of actually counting, we observe a statistical feature of a set (think stream) of elements
- The feature: the number of trailing zeroes of the value of a very good hash function
- We keep track of the maximum $m$ (log log $n$ bits!)
HyperLogLog counters

- Instead of actually counting, we *observe* a statistical feature of a set (think stream) of elements.
- The feature: the number of trailing zeroes of the value of a *very good* hash function.
- We keep track of the maximum $m \ (\log \log n \text{ bits!})$.
- The number of distinct elements $\propto 2^m$. 

HyperLogLog counters

- Instead of actually counting, we observe a statistical feature of a set (think stream) of elements
- The feature: the number of trailing zeroes of the value of a very good hash function
- We keep track of the maximum $m$ (log log $n$ bits!)
- The number of distinct elements $\propto 2^m$
- **Important:** the counter of stream $AB$ is simply
Many, many counters...
Many, many counters...

- To increase confidence, we need several counters (usually $2^b$, $b \geq 4$) and take their harmonic mean
Many, many counters...

- To increase confidence, we need *several* counters (usually $2^b$, $b \geq 4$) and take their harmonic mean

- Thus each set is represented by a list of small (typically 5-bit) counters (unlikely >6 bits!)
Many, many counters...

- To increase confidence, we need several counters (usually $2^b$, $b \geq 4$) and take their harmonic mean
- Thus each set is represented by a list of small (typically 5-bit) counters (unlikely >6 bits!)
- To compute the union of two sets these must be maximized one-by-one
Many, many counters...

- To increase confidence, we need *several* counters (usually $2^b$, $b \geq 4$) and take their harmonic mean

- Thus each set is represented by a list of small (typically 5-bit) counters (unlikely $>6$ bits!)

- To compute the union of two sets these must be maximized one-by-one

- Extracting by shifts, maximizing and putting back by shifts is unbearably slow
Many, many counters...

- To increase confidence, we need several counters (usually $2^b$, $b \geq 4$) and take their harmonic mean.
- Thus each set is represented by a list of small (typically 5-bit) counters (unlikely >6 bits!).
- To compute the union of two sets these must be maximized one-by-one.
- Extracting by shifts, maximizing and putting back by shifts is unbearably slow.
8 bits Broadword!

7 0 2 1

5 3 2 5
8 bits

Broadword!

1 7 1 0 1 2 1 1

0 5 0 3 0 2 0 5
**Broadword!**

8 bits

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

-

<table>
<thead>
<tr>
<th>0</th>
<th>5</th>
<th>0</th>
<th>3</th>
<th>0</th>
<th>2</th>
<th>0</th>
<th>5</th>
</tr>
</thead>
</table>

=
8 bits

\[
\begin{array}{cccccc}
1 & 7 & 1 & 0 & 1 & 2 \\
\hline
0 & 5 & 0 & 3 & 0 & 2 \\
\hline
0 & 125 & 0 & 124 \\
\end{array}
\]
8 bits Broadword!

\[
\begin{array}{ccccccc}
1 & 7 & 1 & 0 & 1 & 2 & 1 & 1 \\
\hline
0 & 5 & 0 & 3 & 0 & 2 & 0 & 5 \\
\hline
1 & 2 & 0 & 125 & 1 & 0 & 0 & 124 \\
\end{array}
\]
8 bits Broadword!

\[
\begin{array}{ccccccc}
1 & 7 & 1 & 0 & 1 & 2 & 1 \\
\end{array}
\]

\[
- \quad - \quad - \\
\begin{array}{ccccccc}
0 & 5 & 0 & 3 & 0 & 2 & 0 & 5 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
1 & 2 & 0 & 125 & 1 & 0 & 0 & 124 \\
\end{array}
\]

\[
\begin{array}{ccccccc}
1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]
8 bits Broadword!

| 1 | 7 | 1 | 0 | 1 | 2 | 1 | 1 |

- 

| 0 | 5 | 0 | 3 | 0 | 2 | 0 | 5 |

= 

| 2 | 125 | 0 | 124 |

| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
**8 bits Broadword!**

```
| 1 | 7 | 1 | 0 | 1 | 2 | 1 | 1 |
```

- 

```
| 0 | 5 | 0 | 3 | 0 | 2 | 0 | 5 |
```

= 

```
| 2 | 125 | 0 | 124 |
```

```
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
```

```
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
```
8 bits Broadword!

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>125</td>
<td>0</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
8 bits Broadword!

<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
</table>

- 

| 0 | 5 | 0 | 3 | 0 | 2 | 0 | 5 |

= 

| 2 | 125 | 0 | 124 |

- 

| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |

- 

| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

= 

| 1 | 0 | 0 | 127 | 1 | 0 | 0 | 127 |
8 bits Broadword!

\[
\begin{array}{cccccccc}
1 & 7 & 1 & 0 & 1 & 2 & 1 & 1 \\
\hline
0 & 5 & 0 & 3 & 0 & 2 & 0 & 5 \\
\hline
2 & 125 & 0 & 124 \\
\hline
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
1 & 0 & 0 & 127 & 1 & 0 & 0 & 127 \\
\end{array}
\]
8 bits Broadword!

\[
\begin{array}{cccccccc}
1 & 7 & 1 & 0 & 1 & 2 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 5 & 0 & 3 & 0 & 2 & 0 & 5 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
2 & 125 & 0 & 124 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
1 & 0 & 0 & 127 & 1 & 0 & 0 & 127 \\
\end{array}
\]
8 bits Broadword!

\[
\begin{array}{cccccccc}
1 & 7 & 1 & 0 & 1 & 2 & 1 & 1 \\
\hline
0 & 5 & 0 & 3 & 0 & 2 & 0 & 5 \\
\hline
2 & 125 & 0 & 124 \\
\hline
1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
\hline
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\hline
1 & 0 & 0 & 127 & 1 & 0 & 0 & 127 \\
\end{array}
\]
<table>
<thead>
<tr>
<th>1</th>
<th>7</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>125</td>
<td>0</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>127</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>127</td>
</tr>
</tbody>
</table>
Other ideas
Other ideas

✦ We keep track of modifications: we do not maximize with unmodified counters
Other ideas

- We keep track of modifications: we do not maximize with unmodified counters
- Systolic computation: each modified set signals back to predecessors that something is going to happen (much fewer updates—\(O(m \log n)\) in expectation! [Cohen])
Other ideas

✦ We keep track of modifications: we do not maximize with unmodified counters

✦ Systolic computation: each modified set signals back to predecessors that something is going to happen (much fewer updates — $O(m \log n)$ in expectation! [Cohen])

✦ Multicore exploitation by decomposition: a task is updating just a batch of counters whose overall outdegree is predicted using an Elias-Fano representation of the cumulative outdegree distribution (almost
Footprint
Footprint

- Scalability: a minimum of 20 bytes per node
Footprint

- Scalability: a minimum of 20 bytes per node
- On a 2TiB machine, 100 billion nodes
Footprint

- Scalability: a minimum of 20 bytes per node
- On a 2TiB machine, 100 billion nodes
- Graph structure is accessed by memory-mapping in a compressed form (WebGraph)
Footprint

- Scalability: a minimum of 20 bytes per node
- On a 2TiB machine, 100 billion nodes
- Graph structure is accessed by memory-mapping in a compressed form (WebGraph)
- Pointer to the graph are store using quasi-succinct lists (Elias-Fano representation)
Performance
Performance

- On a 177K nodes / 2B arcs graph, RSD \(~14\%:\)
Performance

✦ On a 177K nodes / 2B arcs graph, RSD ~14%:

✦ Hadoop: 2875s per iteration [Kang, Papadimitriou, Sun and H. Tong, 2011]
Performance

- On a 177K nodes / 2B arcs graph, RSD ~14%:
  - Hadoop: 2875s per iteration [Kang, Papadimitriou, Sun and H. Tong, 2011]
  - HyperBall on this laptop: 70s per iteration
Performance

- On a 177K nodes / 2B arcs graph, RSD ~14%:
  - Hadoop: 2875s per iteration [Kang, Papadimitriou, Sun and H. Tong, 2011]
  - HyperBall on this laptop: 70s per iteration
  - On a 32-core workstation: 23s per iteration
Performance

- On a 177K nodes / 2B arcs graph, RSD ~14%:
  - Hadoop: 2875s per iteration [Kang, Papadimitriou, Sun and H. Tong, 2011]
  - HyperBall on this laptop: 70s per iteration
  - On a 32-core workstation: 23s per iteration
  - On ClueWeb09 (4.8G nodes, 8G arcs) on a 40-core workstation: 141m (avg. 40s per iteration)
Convergence

Harmonic centrality

![Graph showing relative error versus number of runs](image)
To be fair
To be fair

- Cohen’s estimation framework provides error bounds for the relative error of the probability mass function (and centralities)
To be fair

- Cohen’s estimation framework provides error bounds for the relative error of the probability mass function (and centralities)

- ANF/HyperANF give only pointwise guarantees, but provide error for the *absolute* error of the probability mass function (and centralities)
To be fair

- Cohen’s estimation framework provides error bounds for the relative error of the probability mass function (and centralities)
- ANF/HyperANF give only pointwise guarantees, but provide error for the absolute error of the probability mass function (and centralities)
- Sampling provides only the latter and only for strongly connected graphs
To be fair

- Cohen’s estimation framework provides error bounds for the relative error of the probability mass function (and centralities)
- ANF/HyperANF give only pointwise guarantees, but provide error for the absolute error of the probability mass function (and centralities)
- Sampling provides only the latter and only for strongly connected graphs
- ...but we can retrofit Cohen’s estimators on HyperANF, obtaining an extremely efficient version of Cohen’s framework!
Future Work
Future Work

- Perfect and natural fit for distributed computation (GraphLab, Pregel, etc.)
Future Work

✦ Perfect and natural fit for distributed computation (GraphLab, Pregel, etc.)

✦ Apply the same computational framework to other size estimators
Future Work

✦ Perfect and natural fit for distributed computation (GraphLab, Pregel, etc.)
✦ Apply the same computational framework to other size estimators
✦ Edith Cohen new HIP estimators for HyperLogLogLog counters might work
Future Work

✦ Perfect and natural fit for distributed computation (GraphLab, Pregel, etc.)
✦ Apply the same computational framework to other size estimators
✦ Edith Cohen new HIP estimators for HyperLogLogLog counters might work
✦ [http://webgraph.di.unimi.it/software](http://webgraph.di.unimi.it/software)
Future Work

- Perfect and natural fit for distributed computation (GraphLab, Pregel, etc.)
- Apply the same computational framework to other size estimators
- Edith Cohen new HIP estimators for HyperLogLog counters might work

- [http://webgraph.di.unimi.it/](http://webgraph.di.unimi.it/) ➟ software
- [http://law.di.unimi.it/](http://law.di.unimi.it/) ➟ datasets