When Hashes Met Wedges: A Distributed Algorithm for Finding High Similarity Vectors

Aneesh Sharma
Google

C. Seshadhri
UC Santa Cruz

Ashish Goel
Stanford U
This is based on a true story

The application is real
And now for something completely different...

Big data ➔ Theory ➔ Practice
Here’s an awesome new algorithm for problem P.

2X faster than previous work!

I just bought a bigger cluster with 5X memory.

I’ll just use my old codes.
The problem

• Given $n$ non-negative vectors in $\mathbb{R}^d$, find all “similar” pairs

$$\text{sim}(\mathbf{u}, \mathbf{v}) = \cos \theta$$
(or $$\text{sim}(\mathbf{u}, \mathbf{v}) = \mathbf{u} \cdot \mathbf{v}$$)
• Fundamental for link prediction and recommendation

• [Goel et al 13][Gupta et al 13] Key feature in Who To Follow engine at Twitter
  – Common representations are non-negative
Formally...

• Given $n$ non-negative unit vectors in $\mathbb{R}^d$ and threshold $\tau$, find all pairs $(\mathbf{u}, \mathbf{v})$ such that $\mathbf{u} \cdot \mathbf{v} > \tau$

• In $A^T A$, find all entries $> \tau$
The challenge

I need help!

No existing algorithm works when $n = d = 1B$ and $\text{nnz}(A) > 10B$

There is no systems solution
WHIMP (Wedges and Hashes In Matrix Prod.)

Distributed (MR) algorithm for finding similar vectors

- Theoretically “near-optimal” total shuffle/comm
- Practically viable. Works on \( \text{nnz}(A) = O(100B) \) without killing cluster
The distributed framework

- Synchronous communication along edges (can be simulated in MR)
- Total communication is shuffle cost
Previous art

• Exact matrix mult: [BLAS, Csparse]

• Approx matrix mult, using low rank approximation: [Drineas-Kannan-Mahoney 06] [Sarlos 06][Belabbas-Wolfe 08]

• Random projections, (Asym) LSH [Indyk-Motwani99] [Charikar03] [Andoni-Indyk 06] [Shrivastava-Li15] [Andoni-Indyk-Laarhoven-Razenshteyn-Schmidt15]

• Path sampling: [Cohen-Lewis99] [Schank-Wagner 06][S-Pinar-Kolda 13] [Kolda-Pinar-Plantenga-S-Task 14] [Zadeh-Goel 15] [Ballard-Kolda-Pinar-S 15]
Previous attempts:

- Exact matrix multiplication \[ \text{BLAS, Csparse} \]
- Approximate matrix multiplication, using low-rank approximation: \[ \text{Drineas-Kannan-Mahoney 06, Sarlos 06, Belabbas-Wolfe 08} \]
- Random projections, \( \text{Asym LSH (Indyk-Motwani 99, Charikar 03, Andoni et al. 06, Shrivastava-Li 15)} \)
- Path sampling: \[ \text{Cohen-Lewis 99, Zadeh-Goel 15, Ballard et al.} \]

Too much communication!
Philosophers and psychiatrists should explain why it is that we mathematicians are in the habit of **systematically erasing our footsteps**…

- Gian-Carlo Rota

I’ll tell you about an erased footstep.
The Twitter problem
The Twitter problem

- Users with large intersections of followers tend to be “similar”
The Twitter problem

- Cosine similarity is “normalized intersection”
The Twitter problem

- Domain studies show similarities of 0.15 – 0.2 matter
- 15% of my followers follow you. We need to know
The similarity threshold

- Most literature on low dimensional projections/hashing/nearest neighbor for on sim $> 0.8$
- In recommendations, similarities around 0.1-0.3 matter
# Real recommendations

## Users similar to @www2016ca

<table>
<thead>
<tr>
<th>Rank</th>
<th>Twitter @handle</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>@WSDMSocial</td>
<td>0.268</td>
</tr>
<tr>
<td>2</td>
<td>@WWWfirenze</td>
<td>0.213</td>
</tr>
<tr>
<td>3</td>
<td>@SIGIR2016</td>
<td>0.190</td>
</tr>
<tr>
<td>4</td>
<td>@ecir2016</td>
<td>0.175</td>
</tr>
<tr>
<td>5</td>
<td>@WSDM2015</td>
<td>0.155</td>
</tr>
</tbody>
</table>

## Users similar to @duncanjwatts

<table>
<thead>
<tr>
<th>Rank</th>
<th>Twitter @handle</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>@ladamic</td>
<td>0.287</td>
</tr>
<tr>
<td>2</td>
<td>@davidlazer</td>
<td>0.286</td>
</tr>
<tr>
<td>3</td>
<td>@barabasi</td>
<td>0.284</td>
</tr>
<tr>
<td>4</td>
<td>@jure</td>
<td>0.218</td>
</tr>
<tr>
<td>5</td>
<td>@net_science</td>
<td>0.200</td>
</tr>
</tbody>
</table>
The quadratic bottleneck

- To find similarities of $\tau$, you need $1/\tau^2$ work or communication (or pain)
- A well-engineered solution for $\tau = 0.9$ fails miserably for $\tau = 0.2$ (20X more pain)
Our real contribution

• Theorem: To find similarities of $\tau$, WHIMP requires communication/shuffle

$$(\tau^{-1} \log n) \cdot \text{(\# pairs with sim > } \tau)$$

$$(\tau^{-2} \log n) \cdot (\text{nnz}(A))$$

• In previous methods, the $\tau^{-1}$ and $\tau^{-2}$ terms multiply larger quantities

lower bound on output
typically large
The distributed framework

- Synchronous communication along edges (can be simulated in MR)
- Total communication is shuffle cost
Wedge sampling

- [Cohen-Lewis 99], [Schank-Wagener 06], [S-Pinar-Kolda 13], [Zadeh-Goel 16]
- \(\text{nnz}(A)\) time preprocessing
- In \(O(1)\) time, generates wedge \((i, r, j)\)
- \(\Pr[\text{wedge with ends } i, j] \propto v_i \cdot v_j\)
Wedge sampling

- Weight of path \((i, r, j)\) = \(A_{ri} A_{rj}\)
- Sum over paths from \(i\) to \(j\) = \(\sum_r A_{ri} A_{rj} = v_i \cdot v_j\)
- Sample path proportional to weight; probability of getting \((i, j)\) prop. to \(v_i \cdot v_j\)

– Non-negativity used!
Cohen-Lewis trick

- Preprocess to compute $w_r = \sum_i A_{ri}$
- Build data structure to sample r prop. to $w_r$
Cohen-Lewis trick

- Preprocess to compute $w_r = \Sigma_i A_{ri}$
- Build data structure to sample $r$ prop. to $w_r$
- Pick $i$ w.p. $A_{ri}/w_r$, and repeat to get $j$
- Output $(i,j)$
Wedge sampling

- [Cohen-Lewis 99], [Schank-Wagener 06], [S-Pinar-Kolda 13], [Zadeh-Goel 16]
- \(\text{nnz}(A)\) time preprocessing
- In \(O(1)\) time, generates wedge \((i, r, j)\)
- \(\Pr[\text{wedge with ends } i,j] \text{ proportional to } v_i \cdot v_j\)
Distributed wedge sampling

- [Zadeh-Goel 15] DISCO: Frequent “candidates” tend to be large entries of product matrix
- Requires shuffle/communication of all wedges
Distributed wedge sampling

Frequent "candidates" tend to be large entries of product matrix.

Requires shuffle/communication of all wedges.

So how many wedges do we need to catch all $v_i \cdot v_j > \tau$?
How many wedges?

Pr[wedge with \((i, j)\)] = \frac{\sum_{i,j} v_i \cdot v_j}{\|A^T A\|_1}

Pr[wedge with \(i,j\)] proportional to \(v_i \cdot v_j\)

Sum of all dot products/similarities
How many samples?

$$\Pr[\text{wedge with } (i, j)] = \frac{v_i \cdot v_j}{\|A^T A\|_1}$$

Suppose $$v_i \cdot v_j = \tau$$

$$\# samples \approx \frac{10\|A^T A\|_1}{\tau}$$

We only want large entries in $$A^T A$$

But # wedge samples is linear in $$|A^T A|$$
Signal vs noise

Large $v_i \cdot v_j$  
Small $v_i \cdot v_j$
Signal vs noise

- Too many small entries “drown” out the few large entries
- Most of the communication is noise
How many samples?

\[ \Pr[\text{wedge with } (i, j)] = \frac{v_i \cdot v_j}{\| A^T A \|_1} \]

Suppose \( v_i \cdot v_j = \tau \)

\[ \# \text{samples} \approx \frac{10 \| A^T A \|_1}{\tau} \]
Some numbers

| Dataset   | Dimensions $n = d$ | Size (nnz) | $|A^TA|_1$ | TB shuffle |
|-----------|--------------------|------------|-----------|------------|
| friendster| 65M                | 1.6B       | 7.2E9     | 26.2       |
| clueweb   | 978M               | 42B        | 6.8E10    | 247.4      |
| eu        | 1.1B               | 84B        | 1.9E11    | 691.2      |
| flock     | -                  | $O(100B)$  | 5.1E12    | 18553.7    |

Shuffle = $(10|A^TA|/0.2) \times 16$ bytes

Single round of MR can handle < 150TB

No systems solution for flock
• [Zadeh-Goel 15] DISCO: Frequent “candidates” tend to be large entries of product matrix
• Requires shuffle/communication of all wedges
Pruning with oracle

What is $v_i \cdot v_j$?

$v_i \cdot v_j > \tau$
Pruning with oracle

\[ v_{i'} \cdot v_{j'} < \tau \]

What is \( v_{i'} \cdot v_{j'} ? \)
Eventually

Only wedges with \( v_i \cdot v_j > \tau \) are actually communicated!
But isn’t designing the oracle the problem itself?
A reminder

\(v_1, v_2, v_3, \ldots, v_n\)
If the green node “knows” all the vectors, it can construct the oracle.
But that’s just exact multiplication!
Green node collects “sketches”, and simulates oracle using them.
SimHash [Charikar 03]

- Single bit hash = sign of dot product
- \( \Pr[h(v_i) = h(v_j)] = 1 - \theta/\pi \)
The hashing scheme

- Rinse and repeat $k$ times

$h(v_i) = 1$
$h(v_j) = 0$
The hashing scheme

- Rinse and repeat $k$ times

\[ h(v_i) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \]
\[ h(v_j) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \]
The hashing scheme

- \( \Delta \) is binomial \( B(k, \theta/\pi) \)
  - If \( v_i, v_j \) are orthogonal, \( \Delta \) is \( B(k,1/2) \)
- (Roughly) \( \Delta \approx k\theta/\pi \)
- \( \cos(\pi \Delta/k) \approx \cos(\theta) \)
Choosing the hash length

- [Chernoff bound] Binomial tails
- Require $1/\tau^2$ flips to distinguish
- Need hash of length $1/\tau^2$ to determine similarities around $\tau$
Generating SimHashes

- Sending independent Gaussian for each bit is expensive
- We use pseudorandom seeded Gaussians to reduce communication

\[ h_1(\vec{v}_i) = \vec{v}_i \cdot \vec{g} = \text{sgn} \left( \sum_r M_{i,r} g_r \right) \]
WHIMP = Wedge Sampling + SimHash (Hashes)
• Each processor on right computes $h(v_i)$
• Using pseudorandom generators, $O(\text{nnz}(A))$ communication
WHIMP, Round 2: Getting hashes

- Each vertex on left collects relevant hashes
- All edges send a hash
- Communication = $O(\tau^{-2} \cdot \text{nnz}(A) \cdot \log n)$
WHIMP, Round 3: The Wedges

- Only output wedges that give similar vectors!
- \( \text{Comm} = (\# \tau\text{-similar pairs}) \times (\tau \log n) \)
Some work required

**Theorem 4.1.** Given input matrices $A, B$ and threshold $\tau$, denote the set of index pairs output by WHIMIP algorithm by $S$. Then, fixing parameters $\ell = \lceil c\tau^{-2} \log n \rceil$, $s = (c\log n)/\tau$, and $\sigma = \tau/2$ for a sufficiently large constant $c$, the WHIMIP algorithm has the following properties with probability at least $1 - 1/n^2$:

- **[Recall]** If $(A^TB)_{a,b} \geq \tau$, $(a,b)$ is output.
- **[Precision]** If $(a,b)$ is output, $(A^TB)_{a,b} > \tau/4$.
- The total computation cost is $O(\tau^{-1}\|A^TB\|_1 \log n + \tau^{-2} (\text{nnz}(A) + \text{nnz}(B)) \log n)$.
- The total communication cost is $O(\tau^{-1} \log n \|A^TB\|_{\geq \tau/4} _1 + \tau^{-2} (\text{nnz}(A) + \text{nnz}(B)) \log n + m + n)$.

**Similar pairs output Hashes**

- Careful choice of parameters to get it to work in practice
Evaluations

| Dataset      | Dimensions $n = d$ | Size (nnz) | $|A^T A|_1$  |
|--------------|-------------------|------------|-------------|
| friendster   | 65M               | 1.6B       | 7.2E9       |
| clueweb      | 978M              | 42B        | 6.8E10      |
| eu           | 1.1B              | 84B        | 1.9E11      |
| flock        | -                 | $O(100B)$  | 5.1E12      |

- Hard to validate!
- Stratified sample of vectors by degree (sparsity)
  - 1000 vectors for degree in $[10^i, 10^{i+1}]$
  - Full similarity compute for all of them
Major caveat!

- Prune all high degree vertices on left
  - Removing spammers, or those that follow too many
  - Removes < 5% of edges in real instances
- Removing dimensions that participate in too many vectors
- Reduces skew in communication
Total shuffle: $\tau = 0.2$

<table>
<thead>
<tr>
<th>Dataset</th>
<th>WHIMP (TB)</th>
<th>DISCO est. (TB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>friendster</td>
<td>4.9</td>
<td>26.2</td>
</tr>
<tr>
<td>clueweb</td>
<td>90.1</td>
<td>247.4</td>
</tr>
<tr>
<td>eu</td>
<td>225.0</td>
<td>691.2</td>
</tr>
<tr>
<td>flock</td>
<td>287.0</td>
<td>18553.7</td>
</tr>
</tbody>
</table>

Infeasible to feasible
Communication

<table>
<thead>
<tr>
<th>Dataset</th>
<th>WHIMP (TB)</th>
<th>DISCO est. (TB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>friendster</td>
<td>4.9</td>
<td>26.2</td>
</tr>
<tr>
<td>cluweb</td>
<td>90.1</td>
<td>247.4</td>
</tr>
<tr>
<td>eu</td>
<td>225.0</td>
<td>691.2</td>
</tr>
<tr>
<td>flock</td>
<td>287.0</td>
<td>18553.7</td>
</tr>
</tbody>
</table>

About 9X overhead over optimum

Infeasible to feasible
Precision-recall curves: $\tau = 0.2, 0.4$

- Vary $\sigma$ for precision-recall curves

From hashes, estimate $v_i \cdot v_j$

- $< \sigma$: Ignore
- $> \sigma$: Output
Miles to go before I sleep...
The skew problem?

- Communication to node: $O(d \times \text{hash size})$
  - $O(d \tau^{-1}\log n)$, can be too much

- Alternate scheme to bound max communication?
Minhash alternative?

• 1KB (≈ 8000 bits) sketch barely distinguishes 0 from 0.1

• Better sketches? Even saving ½ in length would be useful
Uses of non-negativity?

- Power of Cohen-Lewis trick
- [Andoni-Razenshteyn 15, 16] Data dependent hashing
  - Using low dimensional structure
Finding large entries in matrix product?

• Find all large entries in product $AB$ (or $A^TA$)
• What is the complexity of this problem?
  – Fine-grained complexity anyone?
Takeaways

• Similarity search/nearest neighbor is extremely relevant when sim values are closer to 0 than 1
  – And it is hard

• WHIMP deals with this regime using wedge sampling and hashing

• Big data required (minor?) rethink
• Systems solutions don’t always work
I DON'T ALWAYS THINK BIG DATA IS INTERESTING

BUT WHEN I DO, IT INVOLVES THEORY
Evaluations

| Dataset  | Dimensions $n = d$ | Size (nnz) | $|A^TA|_1$ |
|----------|-------------------|------------|-----------|
| friendster | 65M               | 1.6B       | 7.2E9     |
| clueweb   | 978M              | 42B        | 6.8E10    |
| eu        | 1.1B              | 84B        | 1.9E11    |
| flock     | -                 | $O(100B)$  | 5.1E12    |

- Hard to validate!
- Stratified sample of vectors by degree (sparsity)
  - 1000 vectors for degree in $[10^i, 10^{i+1}]$
  - Full similarity compute for all of them

- Precision: is everything output similar?
- Recall: does algorithm output all similar pairs?
Per-user results: \( \tau = 0.2 \)

- Accurate for most users
  - Important for recommendation applications
WHIMP, Round 3: The Wedges

- Normally, $\sigma = \tau$
- Vary $\sigma$ for precision-recall curves

From hashes, estimate $v_i \cdot v_j$

- Ignore if $v_i \cdot v_j < \sigma$
- Output if $v_i \cdot v_j > \sigma$