The “Who-To-Follow” System at Twitter: Algorithms, Impact, and Further Research
Twitter is a new way to create, distribute and discover content
just setting up my twttr
Tweets are envelopes of information
Twitter is realtime

Sohaib Athar
@ReallyVirtual

Helicopter hovering above Abbottabad at 1AM (is a rare event).

David Eun
@Eunner

I just crash landed at SFO. Tail ripped off. Most everyone seems fine. I'm ok. Surreal... (at @flySFO) [pic] — path.com/p/1lwrZb

Path
Tweets go everywhere
500M Tweets sent every day

241M monthly active users

76% access on mobile

Over 35 languages
Twitter is Public
Twitter is Public

Everyone gets their own personal broadcast radio station

Users choose who to follow
Twitter is Public

Everyone gets their own personal broadcast radio station

Users choose who to follow

A bit like tuning into just your favorite radio stations (except that you can talk back to the station)
But there are 241M of them...
Who-to-follow product
Recommendations are contextual
Recommendations are contextual
And change in real-time
And change in real-time
History

April 2010: a 3-person team starts implementation

July 30, 2010: The system is released to millions of users

2010-today: Multiple revisions of the system. Ongoing active work by a team of many engineers and data scientists
Technical Details
Technical Details

1. Background:
   Some common algorithms
   Collaborative Filtering

2. Deciding on the right algorithms

3. Efficient implementation

Will focus on the initial design and algorithms
The Follow Graph

A directed network with a set $V$ of $N$ nodes and a set $E$ of $K$ edges

- A node for each user
- A directed edge from user $u$ to user $v$ if $u$ follows $v$

Key Idea: Use the Follow graph to determine which recommendations to make for every user. A Network Inference problem.
Follow Graph
Follow Graph

Two nodes for every user, a “consumer” node and a “producer” node
Two nodes for every user, a “consumer” node and a “producer” node

Follow Graph

Consumer-Producer Graph
Collaborative Filtering

To get recommendations for C, compute similarity scores for all consumers, and relevance scores for all producers, with respect to C.

1. Start with $\text{sim}(C) = 1$

2. Propagate similarity scores along graph edges to compute relevance scores, and vice-versa.

Many propagation methods; Often, a linear system of equations.
Collaborative Filter: Love or Money

How should we do this propagation? Two extremes:

**LOVE:** All the similarity score of a consumer X gets transferred to each producer that X follows, and the same in the reverse direction

---
Equivalent to Singular Value Decompositions in the dense graph limit (HITS)

**MONEY:** If X follows d producers, then a fraction $1/d$ of the similarity score of X gets transferred to each producer that X follows (SALSA)
Here, $d_{IN}(y)$ is the number of edges coming into node $y$, and $d_{OUT}(x)$ is the number of edges going out of node $x$. 

Parameter $\alpha$ ensures that $C$ always has high similarity to herself.

Influence propagates through the network for $O(1/\alpha)$ steps; typically $\alpha$ is between 0.1 and 0.3.
Sally is the consumer C
\( \alpha = 0.2 \)

\[
\begin{align*}
\text{sim}(x) &= (1 - \alpha) \sum_{(x,y) \in E} \text{relevance}(y)/d_{IN}(y), \quad \text{if } x \neq C \\
\text{sim}(C) &= \alpha + (1 - \alpha) \sum_{(C,y) \in E} \text{relevance}(C)/d_{IN}(y) \\
\text{relevance}(y) &= \sum_{(x,y) \in E} \text{sim}(x)/d_{OUT}(x) \\
\sum_{x \in V} \text{sim}(x) &= 1
\end{align*}
\]

Here, \( d_{IN}(y) \) is the number of edges coming into node \( y \), and \( d_{OUT}(x) \) is the number of edges going out of node \( x \).
Money, Illustrated

Sally is the consumer $C$

$\alpha = 0.2$

\[
\begin{align*}
\text{sim}(x) &= (1 - \alpha) \sum_{(x,y) \in E} \text{relevance}(y)/d_{IN}(y), \quad \text{if } x \neq C \\
\text{sim}(C) &= \alpha + (1 - \alpha) \sum_{(C,y) \in E} \text{relevance}(C)/d_{IN}(y) \\
\text{relevance}(y) &= \sum_{(x,y) \in E} \text{sim}(x)/d_{OUT}(x) \\
\sum_{x \in V} \text{sim}(x) &= 1
\end{align*}
\]

Here, $d_{IN}(y)$ is the number of edges coming into node $y$, and $d_{OUT}(x)$ is the number of edges going out of node $x$
Money, Illustrated

Sally is the consumer C
\( \alpha = 0.2 \)

In vector notation,
\[
\begin{bmatrix}
  s_1 \\
  s_2 \\
  s_3
\end{bmatrix} =
\begin{bmatrix}
  0.2 \\
  0 \\
  0
\end{bmatrix} + 0.8
\begin{bmatrix}
  0.5 & 0 & 0 \\
  0.5 & 1 & 0.5 \\
  0 & 0 & 0.5
\end{bmatrix}
\begin{bmatrix}
  r_1 \\
  r_2 \\
  r_3
\end{bmatrix}
\]

or
\[
\begin{bmatrix}
  s_1 \\
  s_2 \\
  s_3
\end{bmatrix} =
\begin{bmatrix}
  0.2 \\
  0 \\
  0
\end{bmatrix} + 0.8
\begin{bmatrix}
  0.5 & 0 & 0 \\
  0.5 & 1 & 0.5 \\
  0 & 0 & 0.5
\end{bmatrix}
\begin{bmatrix}
  1 & \frac{1}{3} & 0 \\
  \frac{1}{3} & 1 & 0 \\
  0 & \frac{1}{3} & 1
\end{bmatrix}
\begin{bmatrix}
  s_1 \\
  s_2 \\
  s_3
\end{bmatrix}
\]
Sally is the consumer $C$

$\alpha = 0.2$

relevance(Bob) = 0.59

relevance(Kumar) = 0.15

relevance(Alex) = 0.26

So Alex is the most relevant recommendation among users that Sally is not already following.
Cosine Similarity

Represent each producer as an N-dimensional vector of 0’s and 1’s, depending on whether a consumer is following that producer.

Bob: [1, 1, 0]     Kumar: [0, 1, 0]      Alex: [0, 1, 1]
Cosine Similarity

\[ \text{CosineSim}(y_1, y_2) = \text{The cosine of the angle between the vectors of } y_1 \text{ and } y_2 \]

Bob: \([1, 1, 0]\)     Kumar: \([0, 1, 0]\)      Alex: \([0, 1, 1]\)

\[ \text{CosineSim}(\text{Bob}, \text{Kumar}) = \frac{1}{\sqrt{2}} \]

\[ \text{CosineSim}(\text{Bob}, \text{Alex}) = \frac{1}{2} \]
Cosine Similarity

For consumer C, recommend the producers that have the highest Cosine Similarity with the producers that C is following.

Bob: [1, 1, 0]  Kumar: [0, 1, 0]  Alex: [0, 1, 1]

\[
\text{CosineSim}(\text{Bob}, \text{Kumar}) = \frac{1}{\sqrt{2}}; \text{ recommended to Sally}
\]

\[
\text{CosineSim}(\text{Bob}, \text{Alex}) = \frac{1}{2}
\]
Personalized PageRank

Given a consumer $C$, perform a random walk on the Follow graph. If the walk is at node $v$, then the walk:

- Jumps back to node $C$ with probability $\alpha$
- Follows a random edge out of $v$ with probability $1 - \alpha$

The Personalized PageRank of node $Y$ is the weight of $Y$ in the stationary distribution of this random walk.
Personalized PageRank

Can be computed as a system of linear equations:

\[ \pi = \alpha e_C + (1 - \alpha) \pi A \]

Here, \( \pi \) is the vector of Personalized PageRanks with \( C \) as the client, \( e_C \) is a vector with 1 in the \( C \)-th position and 0’s everywhere, and \( A \) is the adjacency matrix of the follow graph, normalized by out-degree.

Can also be computed using Monte Carlo (simulating the random walk): Need \( O(1/\delta) \) steps to get accuracy \( \delta \)
Two Challenges

**ALGORITHMS**: Without first building a system, how do we determine which algorithm to implement?

**SCALE**: How can we implement a recommendation algorithm to run in 100-500 ms?

- Can’t even iterate over all the edges once in < 10 seconds
- Definitely, can not solve LPs on the original Follow graph
Choosing the Right Algorithms

Recommendation algorithm: Amongst all possible recommended follows, which are going to be accepted \textit{if shown}?

Prediction algorithm: Amongst all possible follows, \textbf{predict} which are going to be created \textit{organically}?

Prediction algorithms can be tested without first building a recommendation system
A Dark Test

Run various algorithms to predict follows, but don’t display the results. Instead, just observe how many of the top predictions get followed organically.

![Bar chart showing number of follows for different algorithms: Love, Cosine, Personalized Pagerank, and Money. The chart compares the number of follows for Top 100 and Top 1000 predictions.](image.png)
Scale: Use a Pruned Graph

1. For each consumer C, compute a small circle of “trusted nodes” or “close friends”

2. Compute a pruned Consumer-Producer graph for C, with only nodes in this “circle of trust” being chosen on the consumer side

3. Run MONEY on this pruned graph

Separate pruned graph for every C

Also reduces spam and increases personalization
Computing Circle of Trust

Computation was done on a cluster of commodity machines

- Follow graph was stored in main memory

For every user C, its circle of trust is ~1000 highest Personalized PageRank nodes

- Estimated using Monte Carlo, run for ~100K steps
- Blazingly fast, since it does not involve inspecting the whole graph
Personalized PageRank Distribution

The absolute stationary mass of any node in C’s circle of trust is large; hence a small number of Monte Carlo steps suffice.

Idealized log-log plot of the top k-th highest Personalized PageRanks from C. The linear decay implies a power-law distribution, i.e. a long tail.
Personalized PageRank Distribution

Idealized log-log plot of the top Personalized PageRanks from C. The linear decay implies a power-law distribution, i.e. a long tail.

The absolute stationary mass of any node in C’s circle of trust is large; hence a small number of Monte Carlo steps suffice.
Running Money

For every consumer C

→ We have a pruned Consumer-Producer graph (more than 10,000X smaller)

→ We solve the system of linear equations on this pruned graph using power iteration
Power Iteration

Set \( \text{sim}(C) = 1; \) all other sims = 0

Repeat \( J \) times:

- Compute relevance scores using sims
- Compute sim scores using relevances

The operation above is a contraction map; hence we get good convergence in

- \( J = O((\log N)/\alpha) \) iterations in theory
- \( J = 3 \) iterations in practice

\[
\begin{align*}
\text{sim}(x) &= (1 - \alpha) \sum_{(x,y) \in E} \text{relevance}(y)/d_{IN}(y), \quad \text{if } x \neq C \\
\text{sim}(C) &= \alpha + (1 - \alpha) \sum_{(C,y) \in E} \text{relevance}(C)/d_{IN}(y) \\
\text{relevance}(y) &= \sum_{(x,y) \in E} \text{sim}(x)/d_{OUT}(x) \\
\sum_{x \in V} \text{sim}(x) &= 1
\end{align*}
\]

Here, \( d_{IN}(y) \) is the number of edges coming into node \( y \), and \( d_{OUT}(x) \) is the number of edges going out of node \( x \)
Putting it All Together

Built our own graph processing library, Cassovary, for computing recommendations

Delivered computation time of under 500ms per user

Computed (and refreshed regularly) recommendations for all Twitter users
Open-sourced much of Cassovary
Ongoing Investment

System’s initial success fueled new strategic use cases: Search, User growth, Revenue

We are on the third generation of the system

- Better scaling, personalization, quality
- Incorporate Real-time signals
- A machine learning framework
- Distributed implementation

[Gupta, Goel, Lin, Sharma, Wang, Zadeh, WWW 2013]
Novel Contributions

Many online services use a “recommendation” feature: Amazon, Netflix, LinkedIn...

Our modeling and solution of the problem as personalized random walks is unique

- Money with a damping factor; Pruning using “circle of trust”
- Use in revenue targeting

Multiple general scientific advances and publications
Impact
Strategic Impact

Creates billions of new follows every year

- More than 1/8 of new follows are directly via the Who-to-Follow module
- More than 15% of active users (> 36 Million users) make at least one follow every month via this module
Costolo credits Ashish Goel, a Stanford University computer science professor and part-time Twitter consultant, with helping the company realize that tweets could double as ads. Goel had an important insight: Everyone on Twitter is a marketer who wants to promote a link, a piece of news, or a personal update. Twitter’s strongest appeal to advertisers was to allow them to pay to add more heft to a standard message. Ads could then flow not only to the central Twitter website but to all of the various Twitter software programs on the Web and on mobile phones, some of which are administered by third-party companies.
Promoted Products

A substantial majority of Twitter’s Revenue comes from its Promoted Products

- Promoted Tweets
- Promoted Accounts
- Promoted Trends (< 10%)

Twitter’s most recent quarterly revenue (Q4, 2013) was > $240M
Promoted Tweets and Promoted Accounts

Who to follow · Refresh · View all

CKM Advisors @CKMAdvisors
Follow
Followed by Utkarsh Srivast... X

girish sastry @girishsastry
Followed by Utkarsh Srivast... X

Shiv Ramamurthi @mogro...
Followed by Stanford Alumna... X

Popular accounts · Find friends

Tweets

1 new Tweet

Aneesh Sharma @aneeshs · 4m
Feeling lucky to be at #analytics2014 with @ashishgoel @johnsirois @pankaj @sgurumur for our #edelmanaward presentation. Go #teamtwitter!
Reply t3 Retweet Favorite More

John Sirois @johnsirois · 5m
Hanging out with @ashishgoel @sgurumur @pankaj @aneeshs #analytics2014. Special thanks to our #edelmanaward coaches John Birge & Carrie Beam
Reply t3 Retweet Favorite More

Followed by Peter Fenton.
NewRelic @newrelic · Mar 11
4 Essential Tips from the Coding CEO. How New Relic CEO Lew Cirne still builds product: blog.newrelic... Promoted by NewRelic
Reply t3 Retweet Favorite More

Who to follow · Refresh · View all
Promoted Tweets and Promoted Accounts
Promoted Tweets and Promoted Accounts
Pioneers in Leveraging Data Science for:

Resource Optimization
Utilize the digital footprint of resources to identify targeted improvement initiatives.

Read More +
Impact on Revenue

The Who-to-follow module provided a natural place to display the Promoted Accounts product.

Initial targeting of Promoted Tweets and Promoted Accounts used the Who-To-follow system and algorithms.
Adapted the Cosine Similarity algorithm

- Calculate all accounts similar-to those of an advertiser. Target followers of these similar accounts.
- Used a different pruning technique for scaling
- Need a symmetric similarity function: Cosine Similarity is symmetric but Money is not
Impact on Revenue

“The Who-To-Follow system was crucial, in a fundamental way, for the Promoted Accounts product, and the Promoted Tweets product also initially used the Who-To-Follow system’s targeting”

– Alex Roetter (VP of Engineering, Revenue)
Scientific Progress

1. Fast Incremental PageRank
2. Fast Personalized PageRank
3. Algorithms for Social Search
4. Fast Cosine Similarity

(Not all at Twitter)
Incremental PageRank

Updates to social graph are made in real-time

- As opposed to a batched crawl process for web search
- Real-time updates to PageRank are important to capture trending events

Goal: Design an algorithm to update PageRank incrementally (i.e. upon an edge arrival)

- t-th edge arrival: Let \((u_t, v_t)\) denote the arriving edge, \(d_t(v)\) denote the out-degree of node \(v\), and \(\pi_t(v)\) its PageRank
Incremental PageRank via Monte Carlo

Start with \( R = O(\log N) \) random walks from every node

At time \( t \), for every random walk through node \( u_t \), re-route it to use the new edge \( (u_t, v_t) \) with probability \( 1/d_t(u_t) \)

\( \implies \) Time/number of network-calls for each re-routing: \( O(1/\alpha) \)

Claim: This faithfully maintains \( R \) random walks after arbitrary edge arrivals

Need the graph and the stored random walks in fast distributed memory
Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order.

Theorem: # of re-routings per arrival goes to 0

- t-th arrival: # of reroutes = O(N R/(α t))
- Total time over M arrivals = O((N R log N)/α²)
- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]
Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order.

Theorem: # of reroutings per arrival goes to 0

- t-th arrival: # of reroutings \( \leq \frac{N R}{\alpha t} \)
- Total time over M arrivals = \( O\left(\frac{N R \log N}{\alpha^2}\right) \)
- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]
Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order.

Theorem: \# of reroutings per arrival goes to 0
- $t$-th arrival: $\# \text{ of reroutes} = O\left(\frac{N \cdot R}{(\alpha \cdot t)}\right)$
- Total time over $M$ arrivals $= O\left(\frac{N \cdot R \cdot \log N}{\alpha^2}\right)$
- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]
Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order.

Theorem: # of re-routings per arrival goes to 0

- $t$-th arrival: # of reroutes = $O\left(\frac{NR}{\alpha t}\right)$

- Total time over $M$ arrivals = $O\left(\frac{NR \log N}{\alpha^2}\right)$

- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]
Incremental PageRank Time

Assume that the edges of the graph are chosen by an adversary, but then presented in random order.

Theorem: # of reroutings per arrival goes to 0

- $t$-th arrival: # of reroutings = $O\left(\frac{N R}{\alpha t}\right)$
- Total time over $M$ arrivals = $O\left(\frac{N R \log N}{\alpha^2}\right)$
- Comparable to doing power iteration/Monte Carlo just once!

[Bahmani, Goel, Chowdhury, VLDB 2010]
Personalized PageRank

Network-based Personalized Search is not yet mature

Missing technical piece: Efficient algorithms for Personalized PageRank Queries

→ Given source $s$ and target $t$, estimate the Personalized PageRank of $t$ for $s$ with high accuracy, if it is greater than $\delta$
Existing Methods for PPR Queries

Monte Carlo uses time $> \frac{1}{\delta}$

“Local Update” uses time $\frac{d}{\delta}$

$[d = \frac{M}{N}$ is the average degree$]$

On Twitter-2010, if $\delta = \frac{\lambda}{n} \approx 10^{-7}$, then

$$\Pr [\pi(s, t) > \delta] = 1\%$$
FAST PPR

We can answer PPR queries in either

- Average time $\tilde{O}(\sqrt{d/\delta})$
- Worst case time $\tilde{O}(\sqrt{d/\delta})$ with $\tilde{O}(\sqrt{d/\delta})$ storage and pre-processing time per node
- Typical values: $\delta \sim 10^{-8}$, $d \sim 100$; results in a $> 100$-fold decrease
Basic Idea

Intuition: The Birthday Paradox

- Do small number of “forward” random walks from $s$
- Do “reverse” PageRank computation from $t$ using Local Update with low accuracy
- Use number of collisions as an estimator
- Need to “catch” a collision just before it happens
Simple Version of FAST PPR

1. Use Local Update to compute estimates $\hat{\pi}(v,t)$ to accuracy $O(\sqrt{\delta})$.

2. Define

   Target Set $\hat{T}_t = \{v \in V : \hat{\pi}(v,t) > \sqrt{\delta}\}$

   Frontier $\hat{F}_t = \{u \in V \setminus \hat{T}_t : (u,v) \in E \text{ for some } v \in \hat{T}_t\}$

3. Take $O\left(\frac{\log(n)}{\sqrt{\delta}}\right)$ Random Walks $\{W_i\}$, terminating each early if it hits $\hat{F}_t$.

   Define
   
   $$X_i = \begin{cases} 
   \hat{\pi}(u,t), & W_i \text{ hits } u \in \hat{F}_t \\
   0, & W_i \text{ does not hit } \hat{F}_t 
   \end{cases}$$

4. Return empirical mean $\{X_i\}$. 
For a uniformly random target node $t$, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}} (\bar{d} + \log(n))\right).$$
For a uniformly random target node $t$, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}} \left( \bar{d} + \log(n) \right) \right).$$
For a uniformly random target node $t$, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}} (\tilde{d} + \log(n))\right).$$

Reverse work (Local Update)  
Forward work (Monte Carlo)
Running Time for Simple Version

For a uniformly random target node $t$, the average per-query running time is

$$O\left(\frac{1}{\sqrt{\delta}} (\tilde{d} + \log(n)) \right).$$

We get final running time of $\tilde{O}(\sqrt{d/\delta})$ by using different accuracies in forward and reverse computation.

We use $\tilde{O}(\sqrt{d/\delta})$ pre-processing/space to go from average to worst case running time.
Experiments

- Admits Distributed Implementation
- Works when source is a set of nodes
- Lower bound of $\frac{1}{\sqrt{\delta}}$
- Open problem: do we need the $\sqrt{d}$?

[Lofgren, Banerjee, Goel, Seshadhri, manuscript, 2014]
ACM CONFERENCE ON ONLINE SOCIAL NETWORKS (COSN'14)
OCTOBER 1-2, 2014
Dublin, Ireland
BACKUP SLIDES
Ongoing evaluation

![Bar chart showing FTR for different methods: SALSA, Pers. PR, Sim(followings), MCM, Closure.]}
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)
Collaborative Filter: Illustration

Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1)
Use a simple propagation method: divide score by 2 and propagate (ignore the client after step 1).