Mining Real Time Stream

U Kang

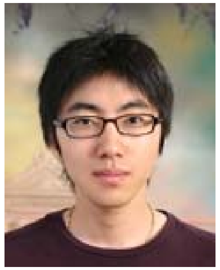
Data Mining Lab
Dept. of Computer Science
KAIST

July 8, 2015

Acknowledgment

Joint work with (Dr.) Yongsub Lim

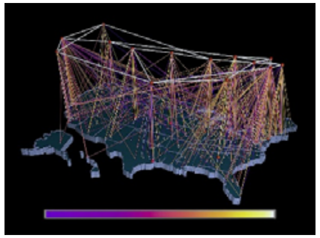
https://sites.google.com/site/ylimhome/



Data Streams are Everywhere

Various real data given (generated) in a stream fashion

0.5B tweets per day



A few Gbps per router

Heartbeat

5T₩ ↑, 0.9B stocks per day

Phone calls

- Click stream
- Query stream in search engine
- Extremely huge data in disk

Data Streams are Everywhere

Various real data given (generated) in a stream fashion



Huge and Fast Growing Data

Stream mining rather than batch analysis

5T₩ ↑, 0.9B stocks per day

Phone calls

- Query stream in search engine
- Extremely huge data in disk

Fundamental Question

How can we analyze stream data in real time?

Desired: Fast, Memory-Efficient, Accurate

Two tasks

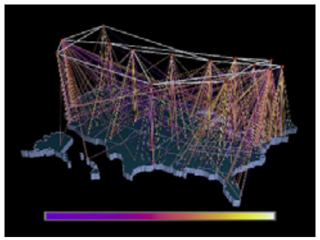
- finding frequent items in data stream
- finding triangles in graph stream

Fundamental Question

Can we track **recently** frequent items in data streams?

"Current" is more important than "past"

Hot keywords?



Active User?

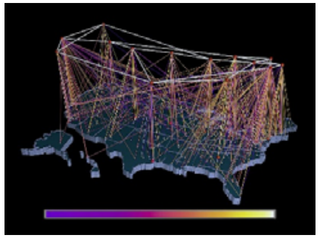
Popular site?

What is Next to Counting Objects?

Counting **relations between objects**

- Which relation in data?

We focus on a **graph**, a set of relations between objects



Clickstream

All can be represented as a graph

Advanced Question

How many relations does each node have? → Degree

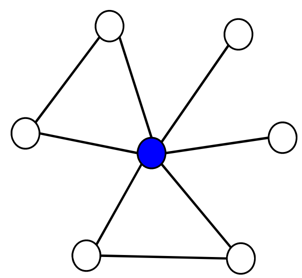
- This is just object (neighbor) counting for each node

How many **tight relations** does each node have?

⬇

How many **triangles** does each node have?

- Local triangle counting problem



Advanced Question

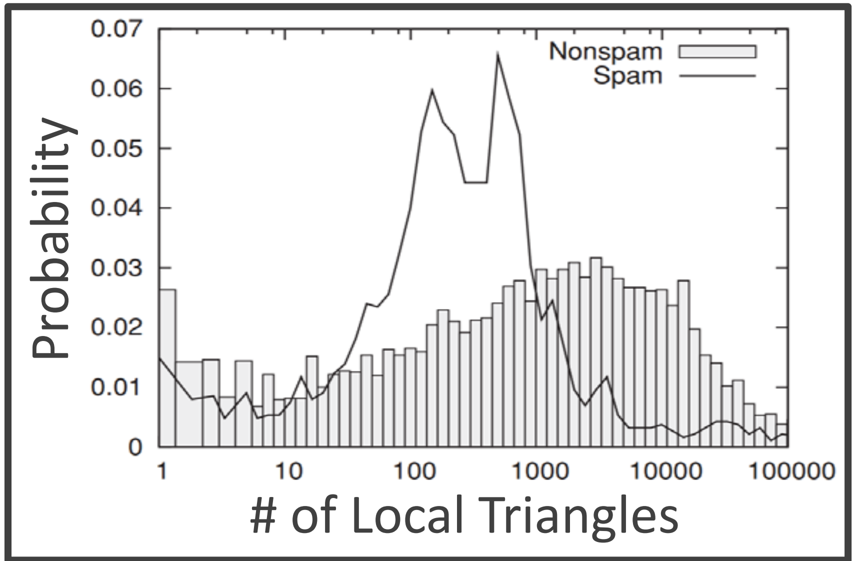
Local Triangle Counting in Graph Streams

of triangles is an **important node feature** in applications

- Node has many triangles
  → it and its neighbors are tightly connected
  → community structure

- Different triangle distribution
  → Spam vs. non-spam pages
  → Fake vs. normal accounts in social networks

- Node feature for anomalous nodes detection in graphs

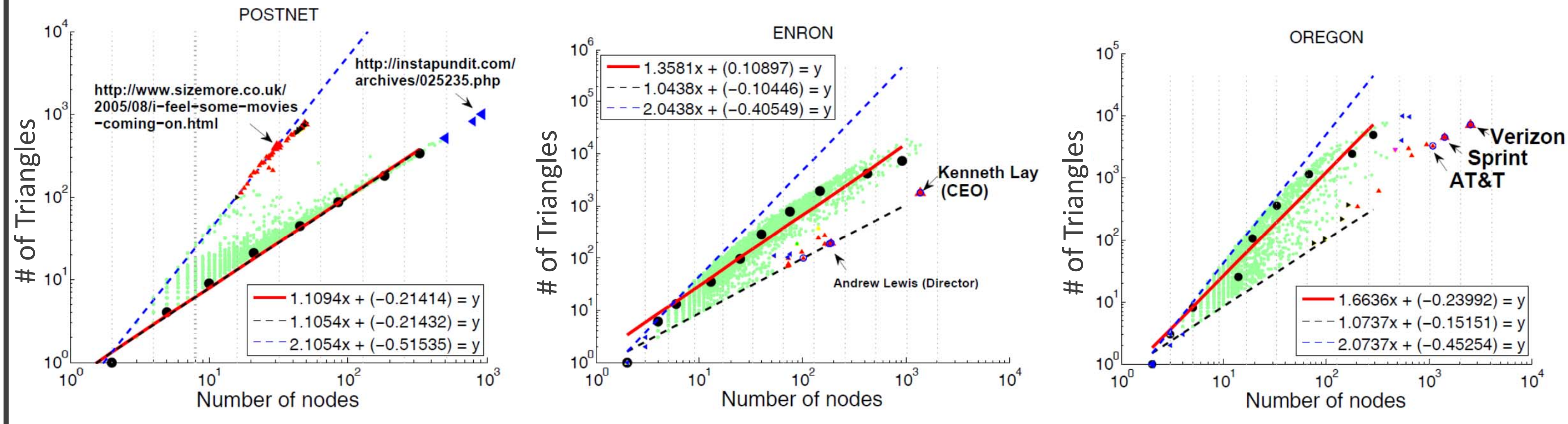


[Becchetti et al., 2010 (TKDD)]

Both Together in Practice

Data-driven anomaly detection in graphs
- Calculate node features
- Examine them in pairwise manner

[Akoglu et al., 2010 (PAKDD)]



(a) *Postnet* (b) *Enron* (c) *Oregon*

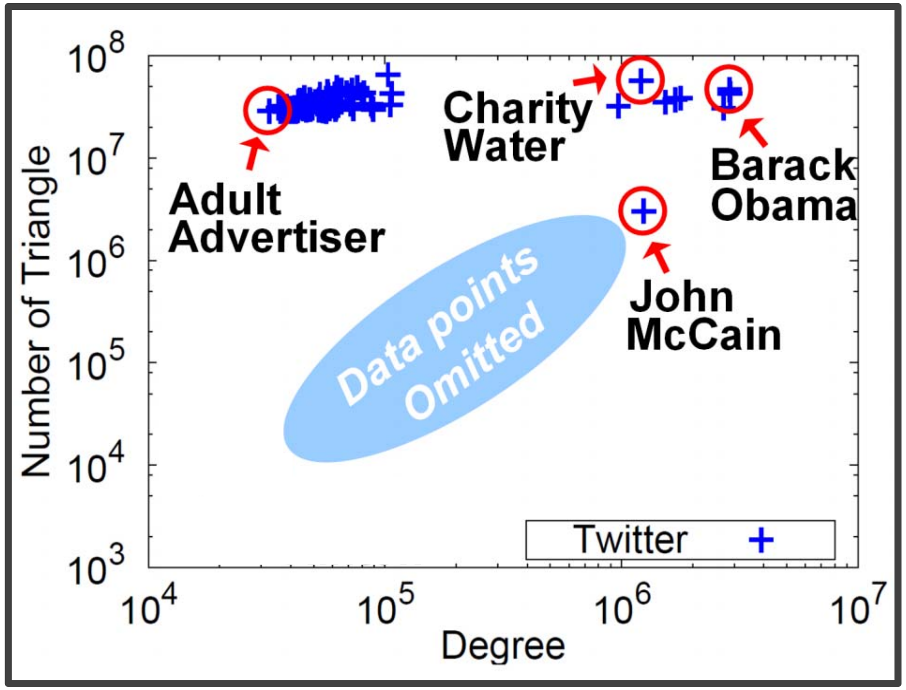
Both Together in Practice

Data-driven anomaly detection in graphs

- Calculate node features
- Examine them in pairwise manner

Previously,
mostly focus on offline analysis

Applying to graph streams is not trivial unless developing online node feature computation



[Kang et al., 2014 (TKDE)]

Both Together in Practice

Data-driven anomaly detection in graphs

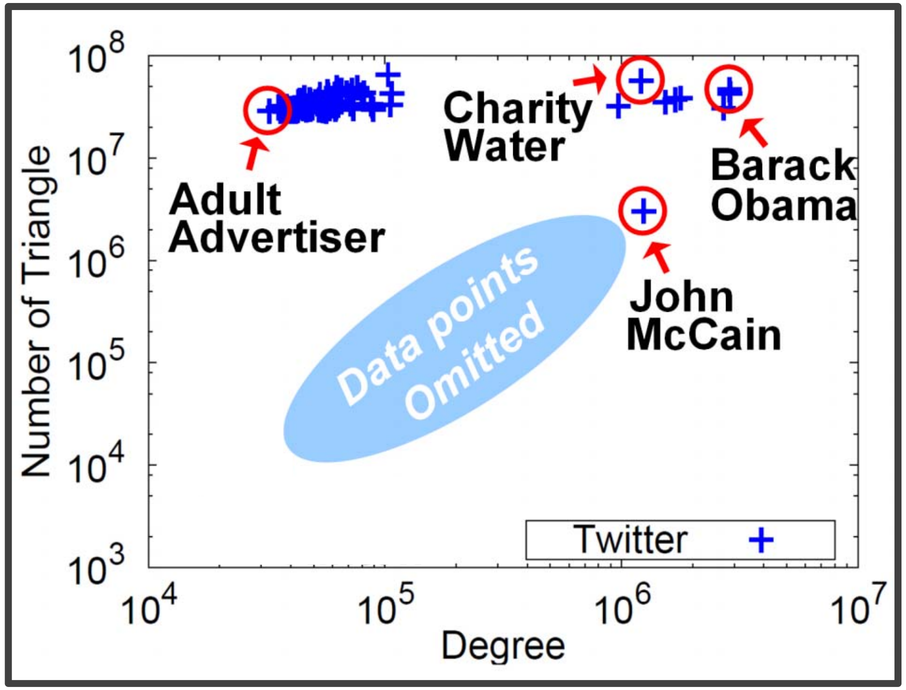
- Calculate node features
- Examine them in pairwise manner

Graph Stream

Recently Frequent Items | Local Triangle Counting

Nodes with **recently active** interaction
Nodes with **tight relations among neighbors**

Real-time Recently Anomalous Node Detection



[Kang et al., 2014 (TKDE)]

Outline

Introduction

<u>Finding Recently Frequent Items in Data Streams</u>

Counting Local Triangles in Graph Streams

Conclusion

Recently Frequent Items

Hot keywords

Messages in SNS

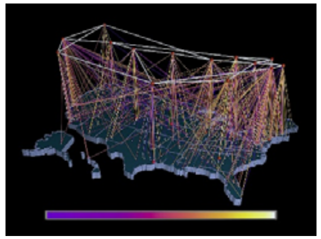
Keywords in search engine

Popularity

Product review

Click stream

Heavy users



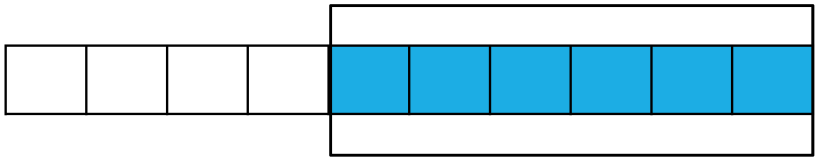
Network Traffic

Problem Definition

Given an item stream, find top-k recently frequent items

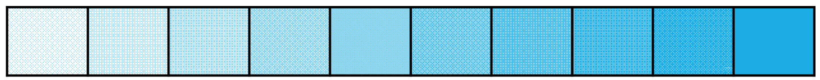
- Item: word, phrase, IP address, product, …
- Frequent: the number of occurrences
- Recentness:

Consider **only past a few items**, and count items



Window based

Count items whose **values decrease over time**

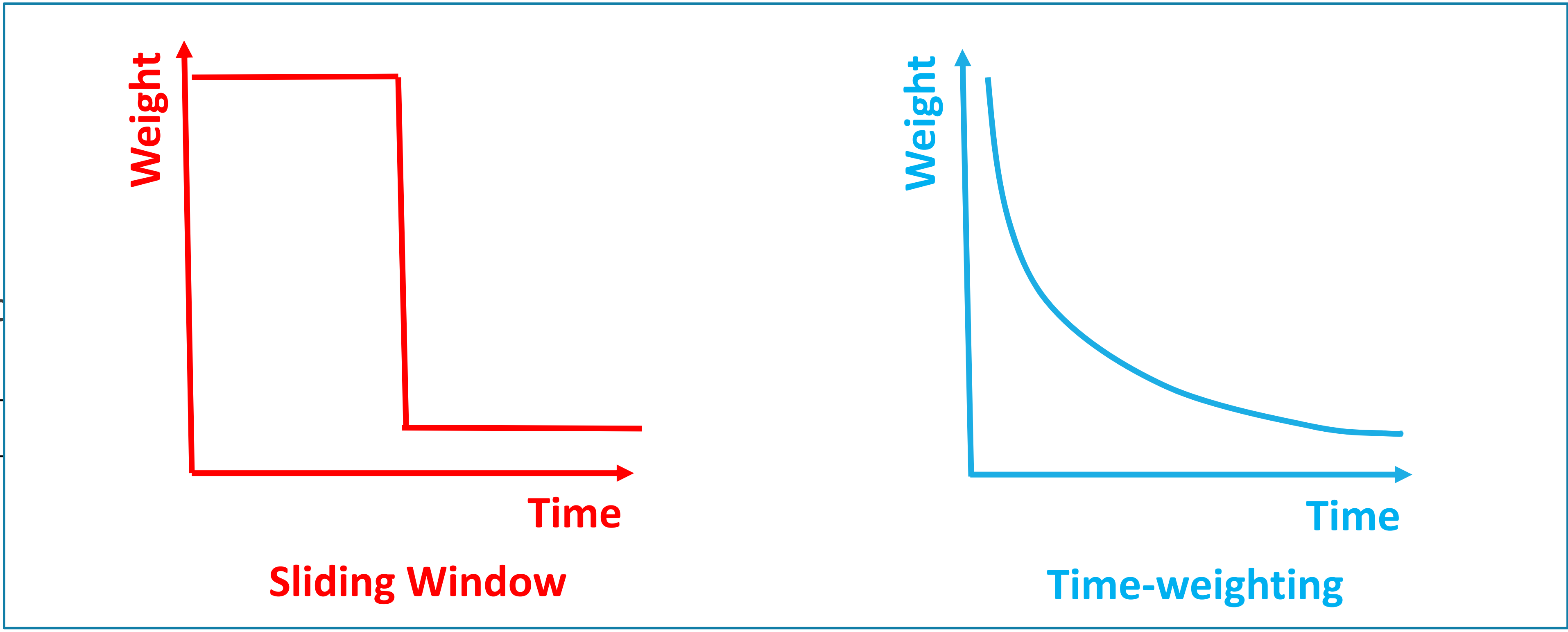


Time-weighting

Problem Definition

Given an item stream, find top-k recently frequent items

Consid



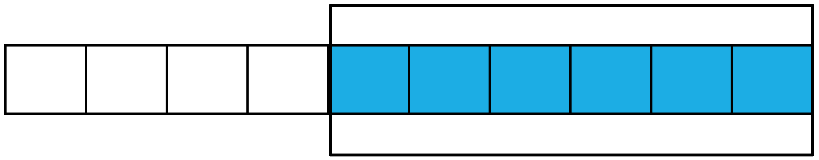
r time

Problem Definition

Given an item stream, find top-k recently frequent items

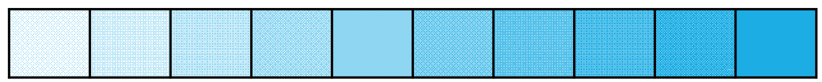
- Item: word, phrase, IP address, product, …
- Frequent: the number of occurrences
- Recentness:

Consider **only past a few items**, and count items



Window based

Count items whose **values decrease over time**



Time-weighting

We choose this approach

Time-weighted Counting

Let t_{cur} be current time

Let item u_t occur at time $t \leq t_{cur}$

	Non Time-weighting	Time-weighting
Weight of Item u_t	1	$\alpha^{t_{cur}-t}$, $(\alpha \leq 1)$

Time-weighted count of item $u = W(u) = \sum_{t \in T(u)} \alpha^{t_{cur}-t}$

- where $T(u)$ is a set of times that u occurred till t_{cur}

Example of Time-weighted Counting

Example. $(\boldsymbol{\alpha = 0.9})$

- 3 distinct items (a, b, c)
- 10 item occurrences

time	1	2	3	4	5	6	7	8	9	10
item	a	a	b	a	c	c	b	b	a	b

At time $t_{cur} = 10$,

$$W(a) = 0.9^{10-1} + 0.9^{10-2} + 0.9^{10-4} + 0.9^{10-9} \approx 2.25$$
$$W(b) = 0.9^{10-3} + 0.9^{10-7} + 0.9^{10-8} + 0.9^{10-10} \approx 3.02$$

Previous Work

Top-k time-weighted frequent item mining in data streams

Sketch based algorithm → large memory spaces

- Count all items approximately using hash table
- Keep top-k [Chen et al., 2014 (Information Science)]

Counter based algorithm → approximation is not good

- Keep buffer, k number of counters
- If buffer overflows, discard minimum item and decrease all counters by the min. count [Zhang et al., 2009 (AICI)]

[Lim et al., 2014 (CIKM)]

Proposed Methods

Two algorithms for top-k time-weighted frequent items

- Both require $O(k)$ memory spaces
  - At most k distinct items are monitored
- **TwSample**: randomized algorithm
  - Exhaustive sampling of item occurrences
  - Theoretical guarantee on accuracy
- **TwMinSwap**: deterministic variation
  - Motivated from TwSample
  - More accurate
  - Fast running time

- Our method (Pro)
- Sketch-based (Sb)
- Counter-based (Cb)

	Best	~	Worst
Memory Usage	Pro	Cb	Sb
Accuracy	Pro	Sb	Cb
Time	Sb	Pro	Cb

[Lim et al., 2014 (CIKM)]

Proposed Methods (TwSample)

Main Idea

Sampling each u_t w.p. $\alpha^{t_{cur}-t}$ using O(k) space

However,

How to control # of distinct sampled items $\leq k$?

Penalized time-weighted count $(\lambda \geq 1)$

$$P(u) = \sum_{t \in T(u)} \boxed{\alpha^{t_{cur}-t+\lambda-1}} = W(u)\alpha^{\lambda-1}$$

Use as sampling probability

[Lim et al., 2014 (CIKM)]

Proposed Methods (TwSample)

time	1	2	3	4	5	6	7	8	9	10
item	a	a	b	a	a	c	b	b	a	b

sampling prob.

$$\alpha^{t_{cur}-t+\lambda-1}$$

$$k = 2$$
$$\lambda = 2$$
$$\sigma = 1$$

Fail
(discarded)

Success
(monitored)

a

Monitored Items

[Lim et al., 2014 (CIKM)]

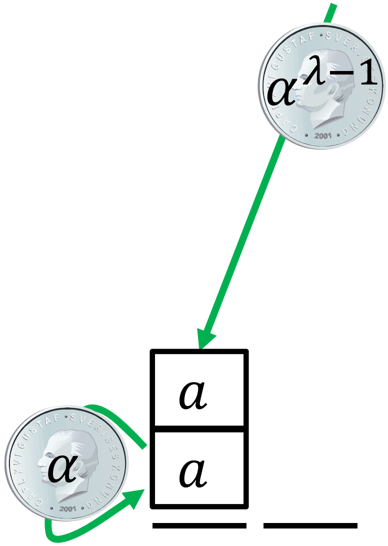
Proposed Methods (TwSample)

time	1	2	3	4	5	6	7	8	9	10
item	a	a	b	a	a	c	b	b	a	b

sampling prob.

$$\boxed{\alpha^{t_{cur}-t+\lambda-1}}$$

$$k = 2$$
$$\lambda = 2$$
$$\sigma = 1$$



Monitored Items

If # of distinct items sampled > k,

- increase λ by σ, (**↓ sampling rate**)
- **re-sample with prob. α^{σ} all** sampled occurrences

Repeat above until # of distinct items sampled $\leq k$

[Lim et al., 2014 (CIKM)]

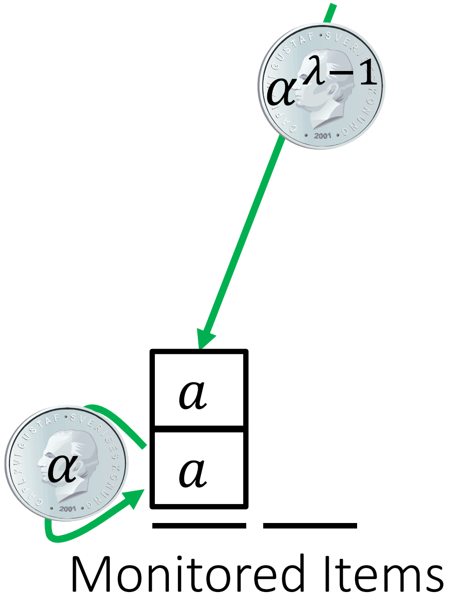
Proposed Methods (TwSample)

time	1	2	3	4	5	6	7	8	9	10
item	a	a	b	a	a	c	b	b	a	b

sampling prob.

$$\alpha^{t_{cur}-t+\lambda-1}$$

$$k = 2$$
$$\lambda = 2$$
$$\sigma = 1$$



At any time t_{cur}, for $u \in K$,

$$\mathbb{E}[c_u] = P(u) = \sum_{t \in T(u)} \alpha^{t_{cur}-t+\lambda-1}$$

[Lim et al., 2014 (CIKM)]

Proposed Methods (TwMinSwap)

TwMinSwap: deterministic variation of TwSample

Main Idea:

Using expected value instead of sampling

Assigning weight $\alpha^{t_{cur}-t}$ to u_t w.p. 1 instead of assigning weight 1 to u_t w.p. $\alpha^{t_{cur}-t}$

[Lim et al., 2014 (CIKM)]

Proposed Methods (TwMinSwap)

Let K be set of items currently counted

Let c_u be counter for item u

For a new item u,

For every $u \in K$, $c_u = \alpha \times c_u$

If $u \in K$	Increase u's count by 1	
If $u \notin K$ & $\lvert K \rvert < k$	$K = K \cup \{u\}$ with $c_u = 1$	
If $u \notin K$ & $\lvert K \rvert = k$,	If $\min_{v \in K} c_v < 1$	Swap min. item with u
	If not	Skip u

weight of new item

[Lim et al., 2014 (CIKM)]

Evaluation Setup

Synthetic: Power-law item distribution

Real:

- **Facebook-wallpost**: who posts whose walls
- **Lastfm-band**: who listens to which bands
- **Lastfm-song**: who listens to which songs
- **DARPA98**: network traffic

Competitors:

- TwFreq1 (counting based)
- TwHCount2 (sketch based)
- SpaceSaving3 (No time-weighting)

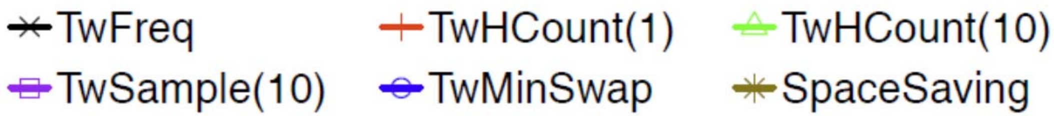
Paramters

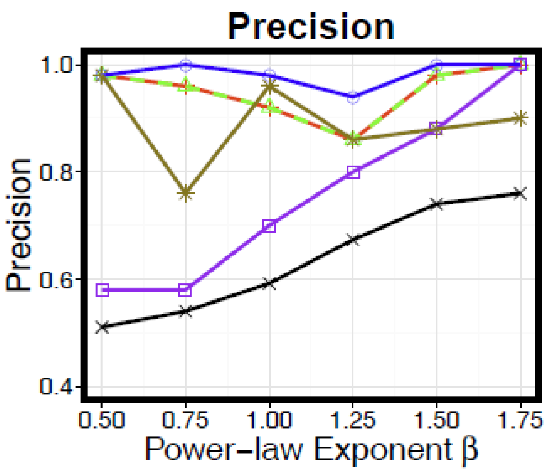
- $k = 50$
- $\alpha = 0.99$

[1] Zhang et al., 2009
[2] Chen and Mei, 2014
[3] Metwally et al., 2005

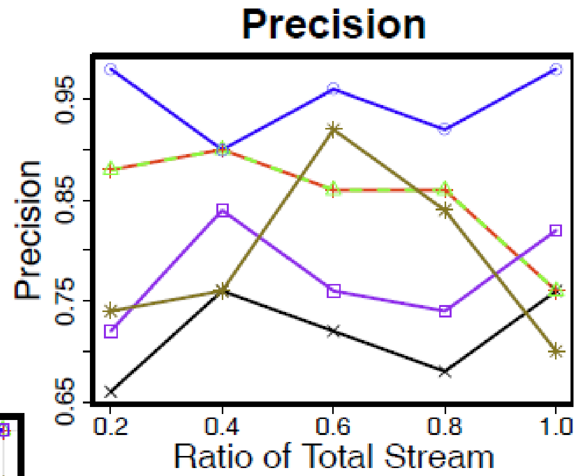
[Lim et al., 2014 (CIKM)]

Precision

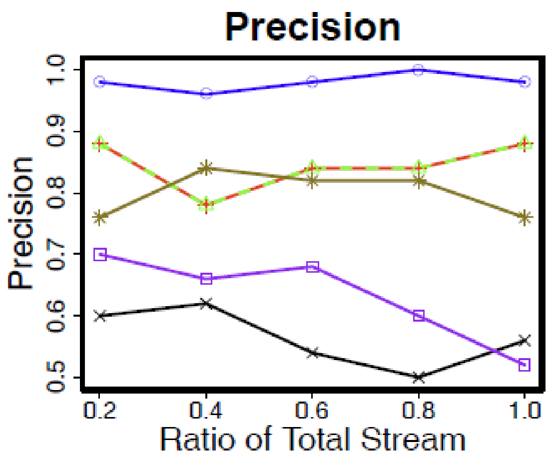




Synthetic Data

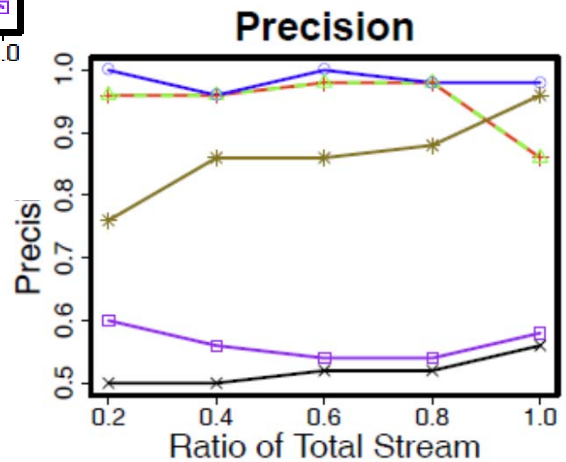


(d) Lastfm-Band



(a) Facebook-Wallpost

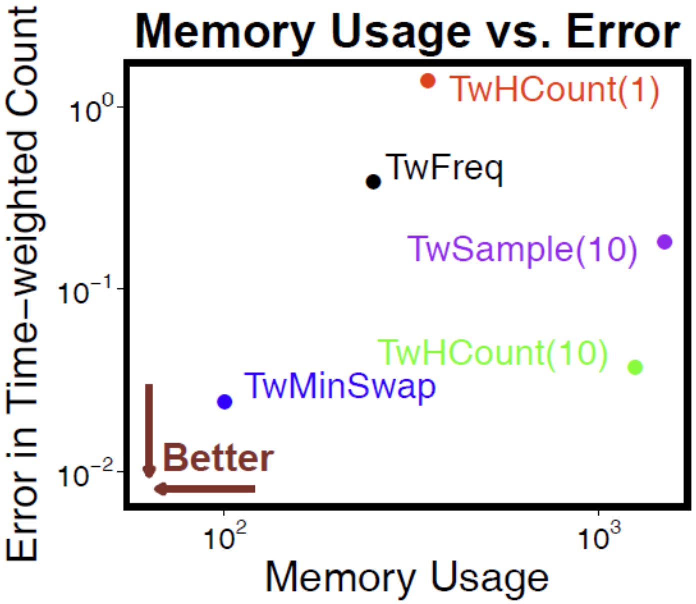
TwMinSwap is best for all cases



(g) Lastfm-Song

[Lim et al., 2014 (CIKM)]

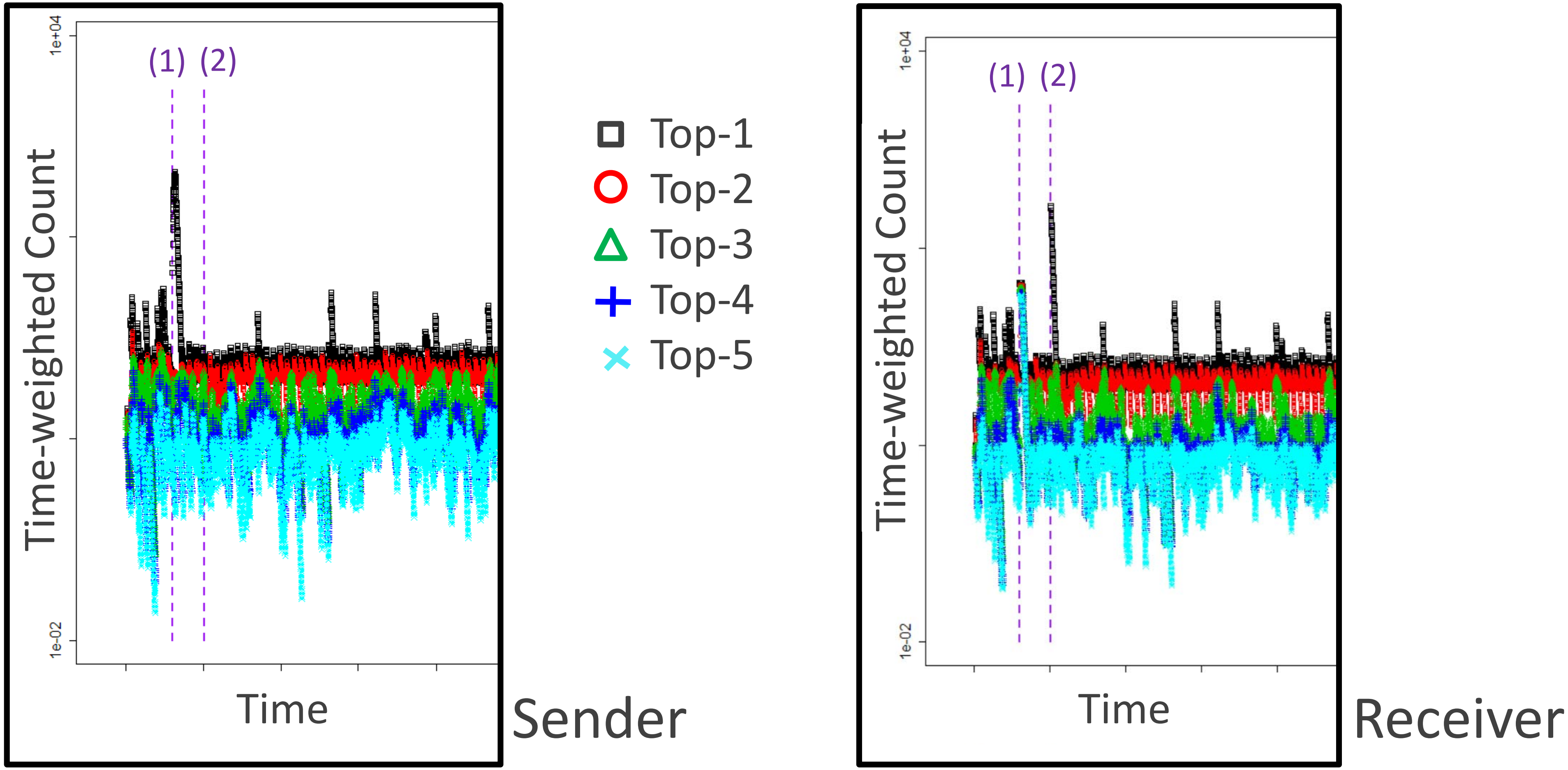
Error vs. Memory Usage



TwMinSwap is most accurate with smallest memory !

[Lim et al., 2014 (CIKM)]

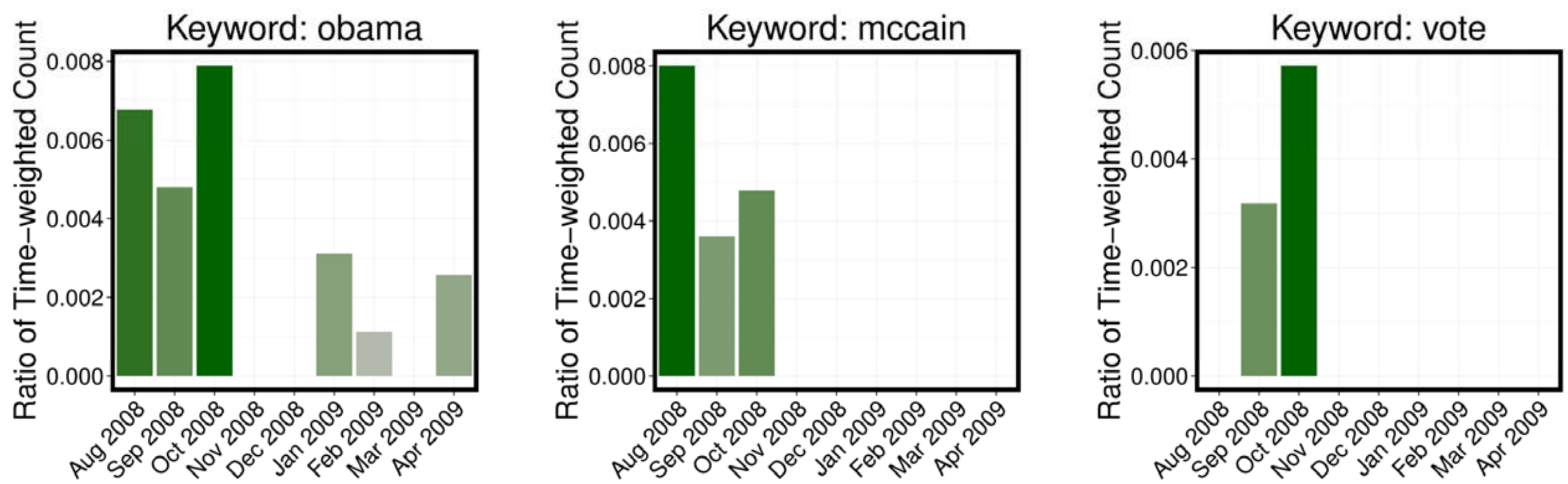
Detecting Network Attacks (DARPA98)



[Lim et al., 2014 (CIKM)]

Discovery in Online Media

MemeTracker (Leskovec et al., 2009)	
Description	Phrases from blogs and news media with timestamps



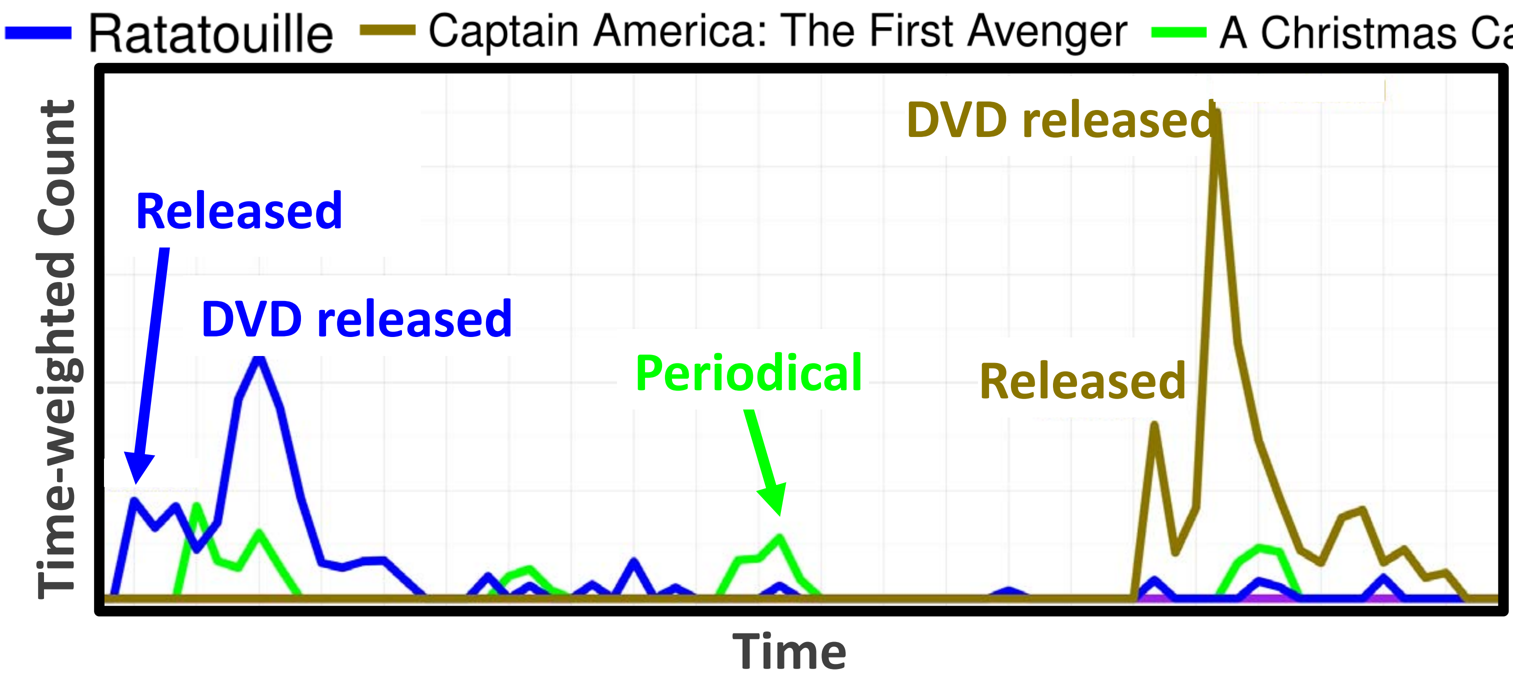
(a) U.S. presidential election

Words related to a certain event become frequent on event time

[Lim et al., 2014 (CIKM)]

Discovery in Movie Reviews

Amazon Movie Reviews (McAuley and Leskovec, 2013)	
Description	Movie reviews



1. Doubly active pattern
2. Periodic pattern

Outline

Introduction

Finding Recently Frequent Items in Data Streams

<u>Counting Local Triangles in Graph Streams</u>

Conclusion

Triangle as Important Node Features

Spam detection	[Becchetti et al., 2010]
Fake account detection	[Yang et al., 2011]
Community detection	[Berry et al., 2011]
Social role identification	[Welser et al., 2007]
Anomaly detection	[Akoglu et al., 2010]

…

Trian

Spam d

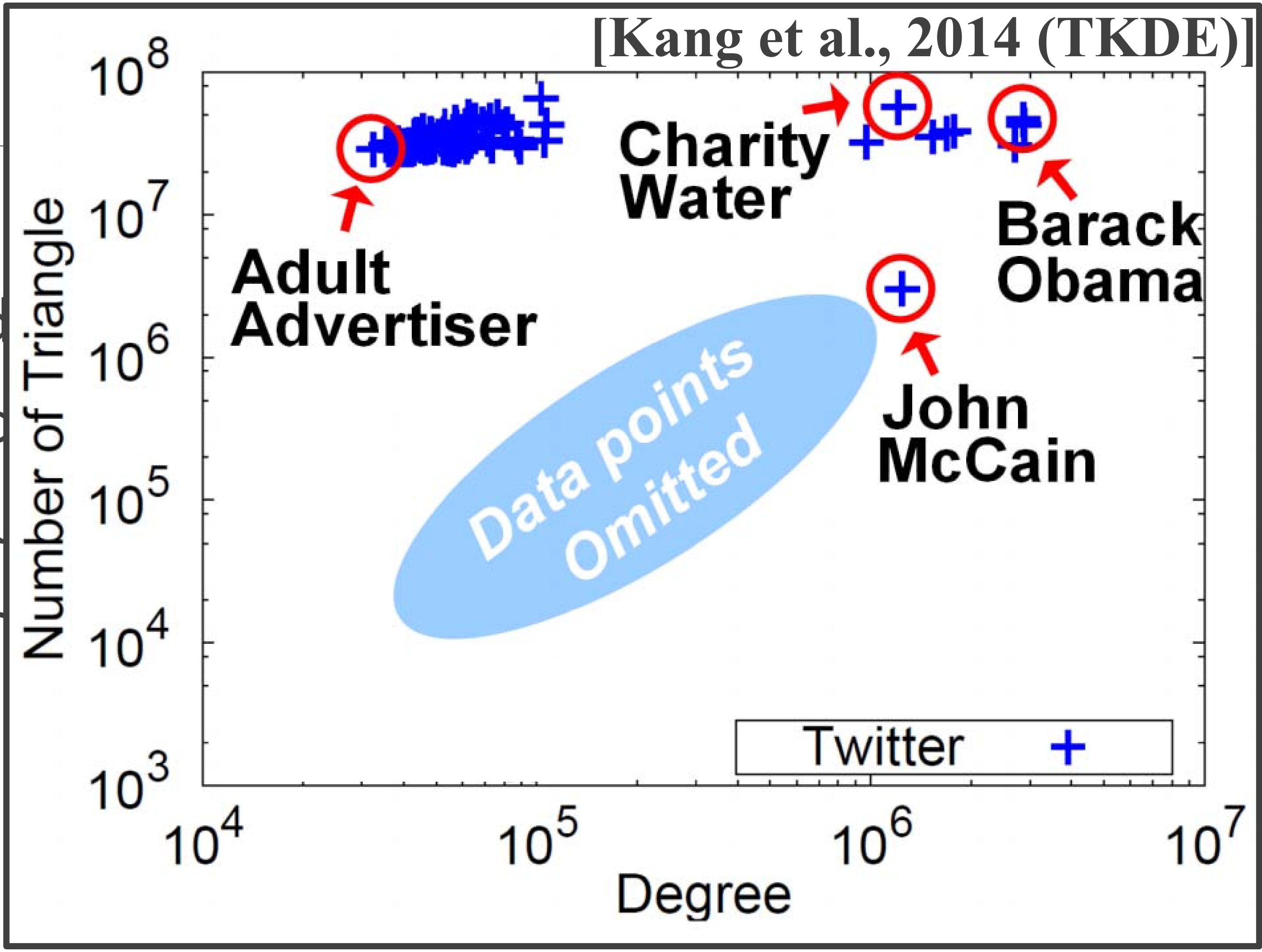
Fake ac

Commu

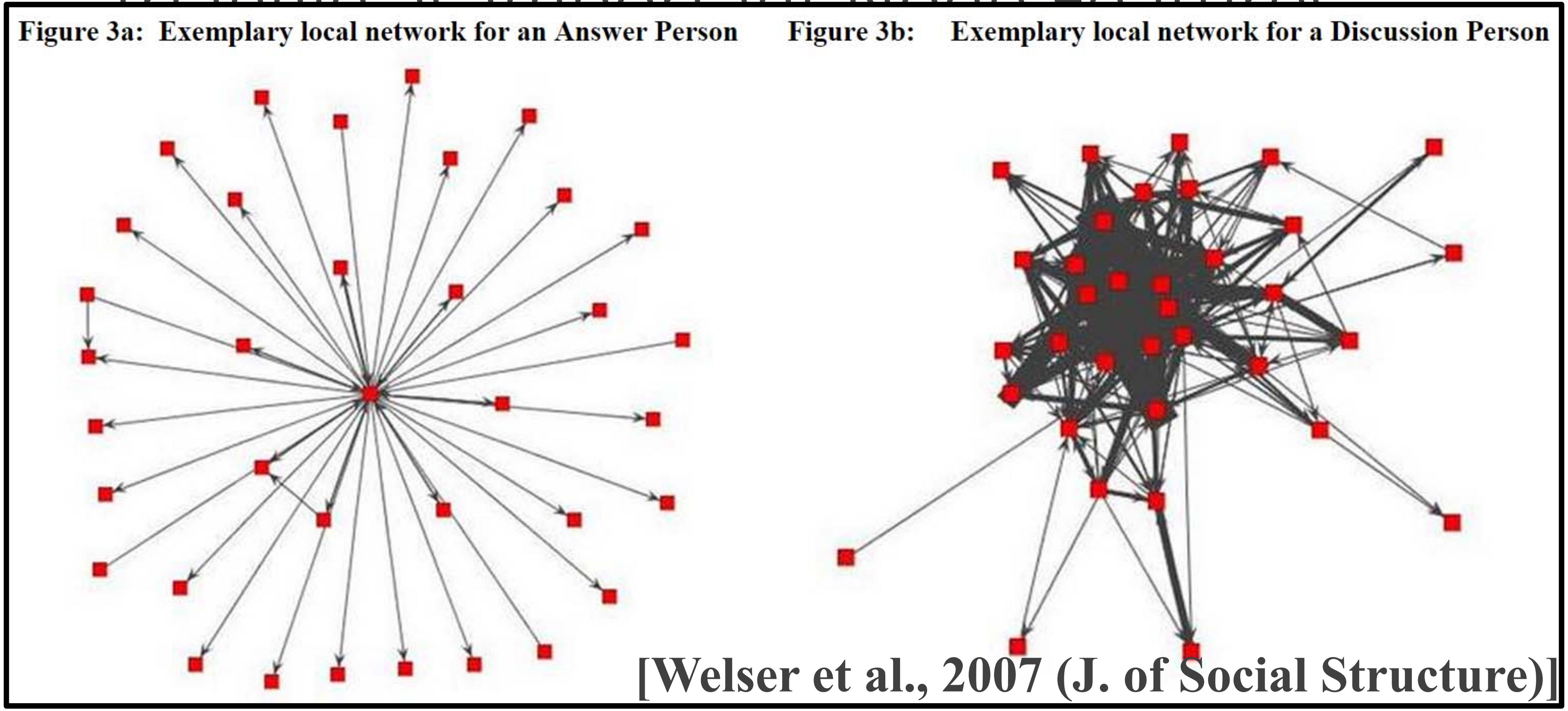
Social r

Anoma

…

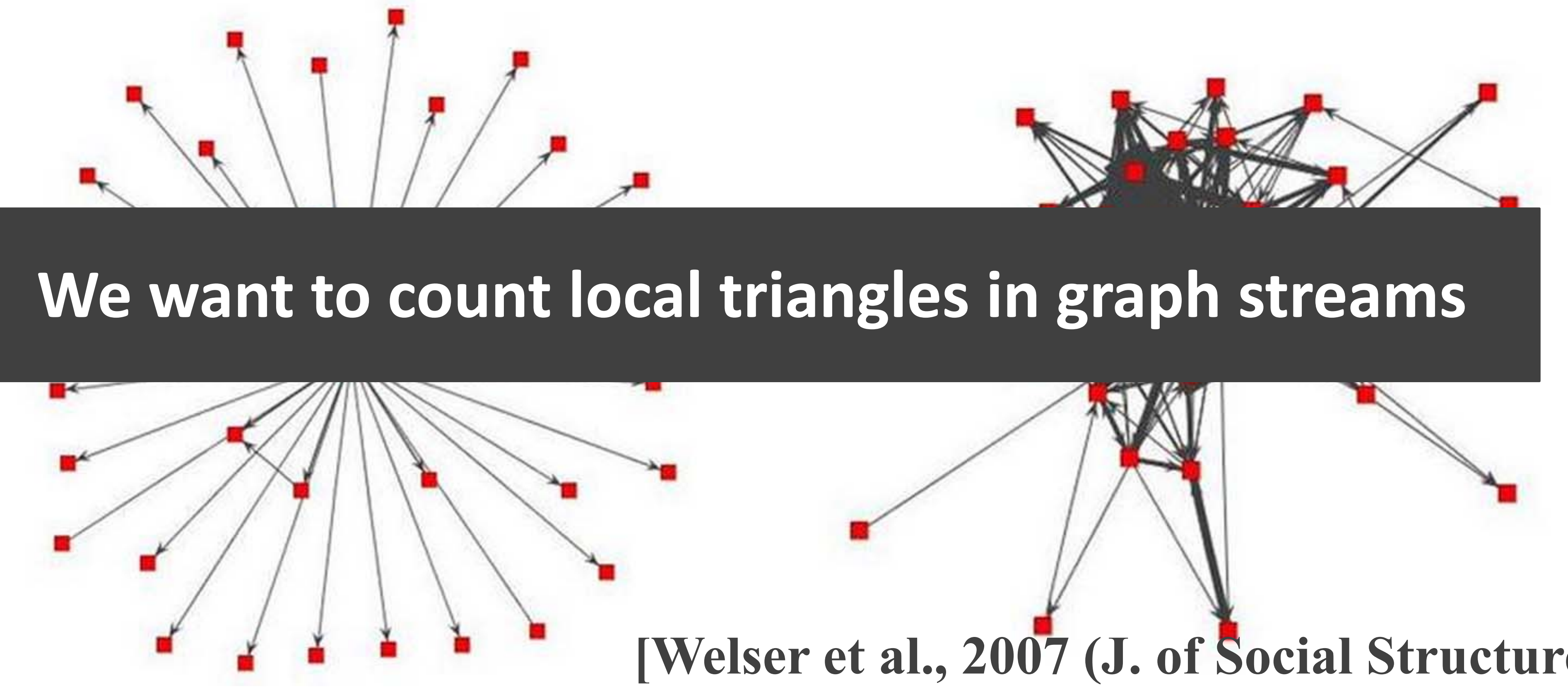


Triangle as Important Node Features



Triangle as Important Node Features

Figure 3a: Exemplary local network for an Answer Person Figure 3b: Exemplary local network for a Discussion Person



We want to count local triangles in graph streams

[Welser et al., 2007 (J. of Social Structure)]

Problem Definition

Given a graph stream, count # of local triangles for all nodes

- Graph stream is a data stream whose item is an edge
- Triangle in graph: cycle of length 3

Essentially, high complexity (super linear)

Previous Work

KP [Kutzkov and Pagh, 2013 (WSDM)]

- Given graph stream, generate K independently sampled graphs
- Count local triangle for each sampled graph and merge the results

Rather theoretic

- Many parameters (may require # of nodes of the graph)
- Not enough accuracy
- Approximation is only for high degree nodes
- Not incremental

[Lim et al., 2015 (KDD)]

Our Approaches

Sampling edges

- For memory efficiency

→ **How to sample?**

Count for sampled graphs → **Underestimated definitely**

Estimate true values → **How to get estimation?**

[Lim et al., 2015 (KDD)]

Proposed Method (MASCOT)

MASCOT: Memory-efficient and Accurate Sampling for Counting Local Triangles in Graph Streams

MASCOT-C
(MASCOT with constant sampling)

MASCOT-A
(MASCOT with adaptive sampling)

Memory-efficiency

Accuracy

MASCOT

[Lim et al., 2015 (KDD)]

Proposed Method (MASCOT)

MASCOT: Memory-efficient and Accurate Sampling for

	[Proposed]	Basic		Existing
	MASCOT	MASCOT-A	MASCOT-C	KP [21]
Corr. Coef.	**High**	**High**	Medium	Low
Error	**Small**	Medium	Medium	Large
Memory Usage	**Low**	Medium	Medium	Large
Estimated Node Range	**All**	**All**	**All**	Top-k
Incremental Update	**Yes**	**Yes**	**Yes**	No

MASCOT

[Lim et al., 2015 (KDD)]

Overviews of Algorithms

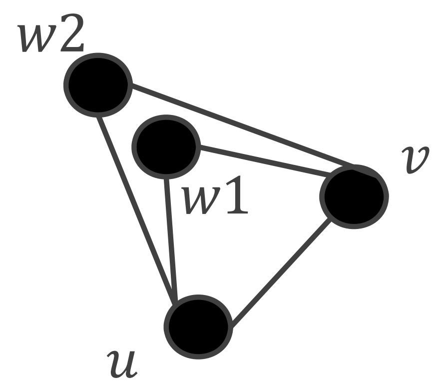
Whenever new edge $e = (u, v)$ arrives
- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

This is for a sampled graph
How to calc. estimation for original graph?

Let $N = N_u \cap N_v$

\# of triangles of $w \in N$ increases by 1

\# of triangles of each of u and v increases by $|N|$



[Lim et al., 2015 (KDD)]

MASCOT-C

Whenever new edge $e = (u, v)$ arrives

- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT-C

- $q = p$ (p is constant user-defined parameter)
- $\tau_w += 1/p^3$
- $\tau_u += |N|/p^3$
- $\tau_v += |N|/p^3$

[Lim et al., 2015 (KDD)]

MASCOT-C

Whenever new edge $e = (u, v)$ arrives

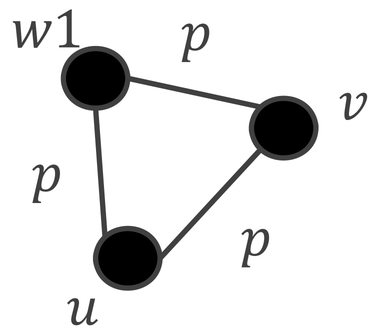
- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT-C

- $q = p$ (p is constant user-defined parameter)
- τ_w += $1/p^3$
- τ_u += $|N|/p^3$
- τ_v += $|N|/p^3$



$\Delta_u = \mathbb{E}[\boldsymbol{\tau_u}]$

for every node $\boldsymbol{u}$

[Lim et al., 2015 (KDD)]

MASCOT-C

Whenever new edge $e = (u, v)$ arrives

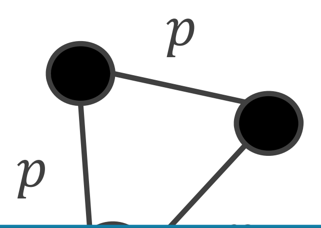
- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT-C

- $q = p$ (p is constant user-defined parameter)



PROS: Memory efficiency (can be estimated)

CONS: Large variance

[Lim et al., 2015 (KDD)]

MASCOT-A

Whenever new edge $e = (u, v)$ arrives

- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT-A

- $q = 1$ if e forms a triangle; otherwise $q = p$
- $\tau_w \mathrel{+}= 1/q_{uv}q_{vw}q_{wu}$
- $\tau_u \mathrel{+}= \sum_{w \in N} 1/q_{uv}q_{vw}q_{wu}$
- $\tau_v \mathrel{+}= \sum_{w \in N} 1/q_{uv}q_{vw}q_{wu}$

[Lim et al., 2015 (KDD)]

MASCOT-A

Whenever new edge $e = (u, v)$ arrives

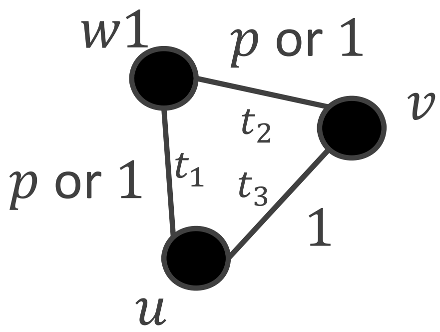
- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT-A

- $q = 1$ if e forms a triangle; otherwise $q = p$
- τ_w += $1/q_{uv}q_{vw}q_{wu}$
- τ_u += $\sum_{w \in N} 1/q_{uv}q_{vw}q_{wu}$
- τ_v += $\sum_{w \in N} 1/q_{uv}q_{vw}q_{wu}$



$\Delta_u = \mathbb{E}[\tau_u]$

for every node u

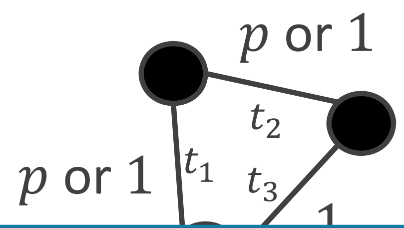
[Lim et al., 2015 (KDD)]

MASCOT-A

Whenever new edge $e = (u, v)$ arrives
- Sample e with probability q
- If e is sampled, for every node, update its triangle counts

Let $N = N_u \cap N_v$
Let $w \in N$

Details for MASCOT-A
- $q = 1$ if e forms a triangle; otherwise $q = p$



PROS: variance is small than MASCOT-C

CONS: more spaces (undesirably)

[Lim et al., 2015 (KDD)]

MASCOT (Main Proposed Algorithm)

Whenever new edge $e = (u, v)$ arrives

- Sample e with probability q
- ~~If e is sampled~~, for every node, update its triangle counts

Let $N = N_u \cap N_v$

Let $w \in N$

Main Idea (Decoupling counting and sampling)

- **Counting and sampling are independently done to each other**

[Lim et al., 2015 (KDD)]

MASCOT (Main Proposed Algorithm)

Whenever new edge $e = (u, v)$ arrives

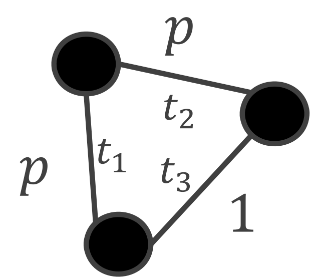
- For every node, update its triangle counts
- Sample e with probability q

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT

- $q = p$
- $\tau_w \mathrel{+}= 1/p^2$
- $\tau_u \mathrel{+}= |N|/p^2$
- $\tau_v \mathrel{+}= |N|/p^2$



$\Delta_u = \mathbb{E}[\tau_u]$

for every node u

[Lim et al., 2015 (KDD)]

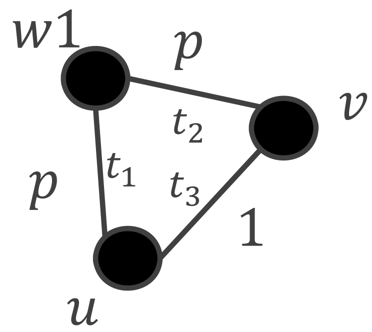
MASCOT (Main Proposed Algorithm)

Whenever new edge $e = (u, v)$ arrives
- For every node, update its triangle counts
- Sample e with probability q

Let $N = N_u \cap N_v$

Let $w \in N$

Details for MASCOT
- $q = p$
- $\tau_w \mathrel{+}= 1/p^2$
- $\tau_u \mathrel{+}= |N|/p^2$
- $\tau_v \mathrel{+}= |N|/p^2$



$\Delta_u = \mathbb{E}[\tau_u]$

for every node u

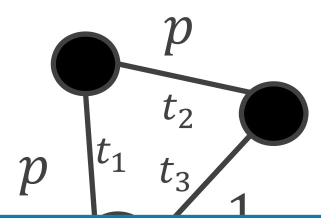
[Lim et al., 2015 (KDD)]

MASCOT (Main Proposed Algorithm)

Whenever new edge $e = (u, v)$ arrives
- For every node, update its triangle counts
- Sample e with probability q

Let $N = N_u \cap N_v$
Let $w \in N$

Details for MASCOT
- $q = p$



Achieve both
1) **Space requirement** (as of MASCOT-C)
2) **Variance** (same upper bound as MASCOT-A)

[Lim et al., 2015 (KDD)]

Evaluation Setting

Three evaluation metrics

- Pearson correlation coefficient
- Error in local triangle counts
- # of sampled edges

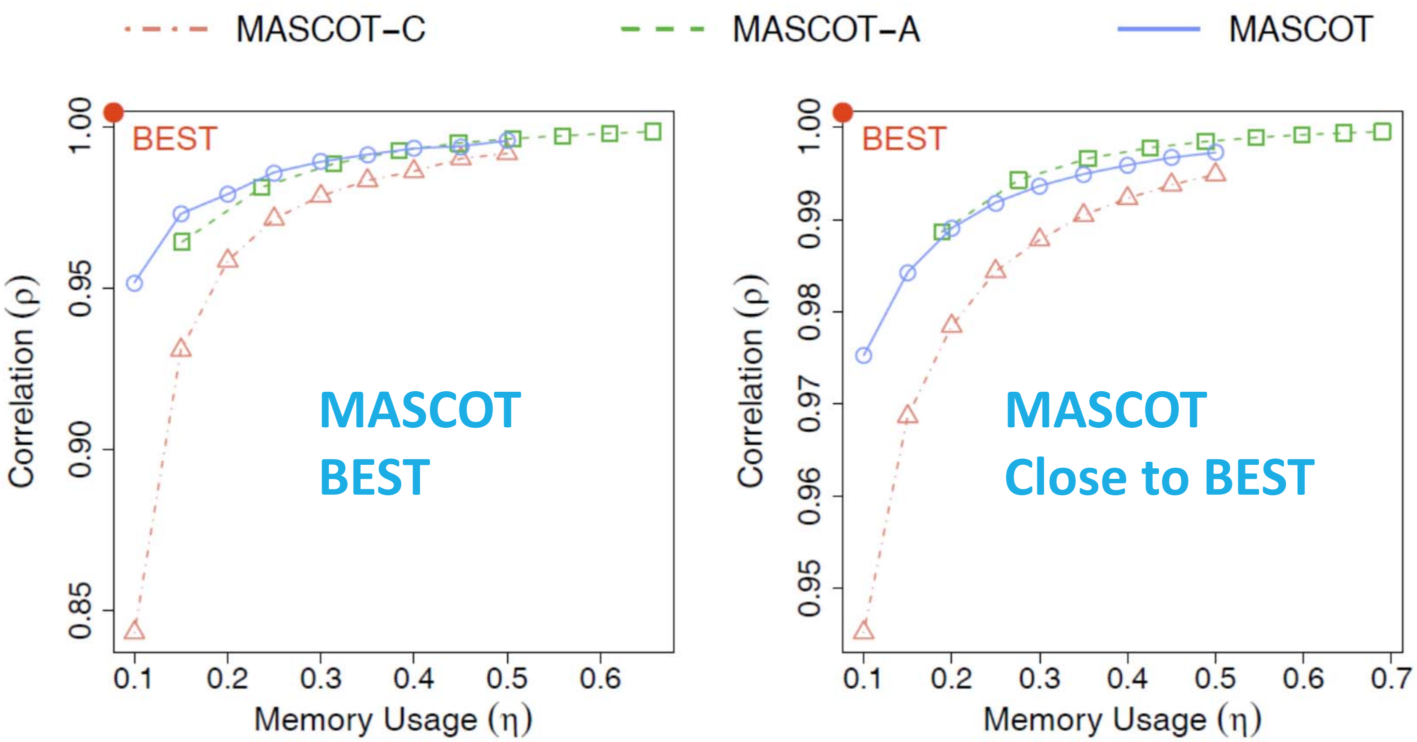
Competitors

- [Kutzkov et al., 2013]

Name	Nodes	Edges	Description
Advogato[1]	5,155	39,285	Trust network
Java-Jung[1]	6,120	50,290	Software class dependency
Oregon[2]	11,461	32,730	Router connections
Google[1]	15,763	148,585	Google internal hyperlinks
CondMat[2]	23,133	93,439	Collaboration network
Cit-Hepph[2]	34,546	420,877	Paper citation network
Enron[2]	36,692	183,831	Enron email exchanges
Slashdot[1]	51,083	116,573	Replay network
Facebook[1]	63,731	817,035	Friendship network
Epinions[2]	75,879	405,740	Trust network
Wiki-Conflict[1]	116,836	2,027,871	Edit confliction
Wordnet[1]	146,005	656,999	Word association network
Gowalla[2]	196,591	950,327	Online social network
Stanford[2]	281,903	1,992,636	Web graph of Stanford.edu
NotreDame[2]	325,729	1,090,108	Web graph of Notre Dame
Amazon[2]	334,863	925,872	Co-purchasing network
BerkStan[2]	685,230	6,649,470	Web graph of Berkeley and Stanford
LiveJournal[2]	4,846,609	42,851,237	LiveJournal online social network
MovieRev9[2]	253,045	6,611,899	Co-reviewed movies in Amazon
DBLP[1]	1,314,050	5,362,414	Co-author network in DBLP

[Lim et al., 2015 (KDD)]

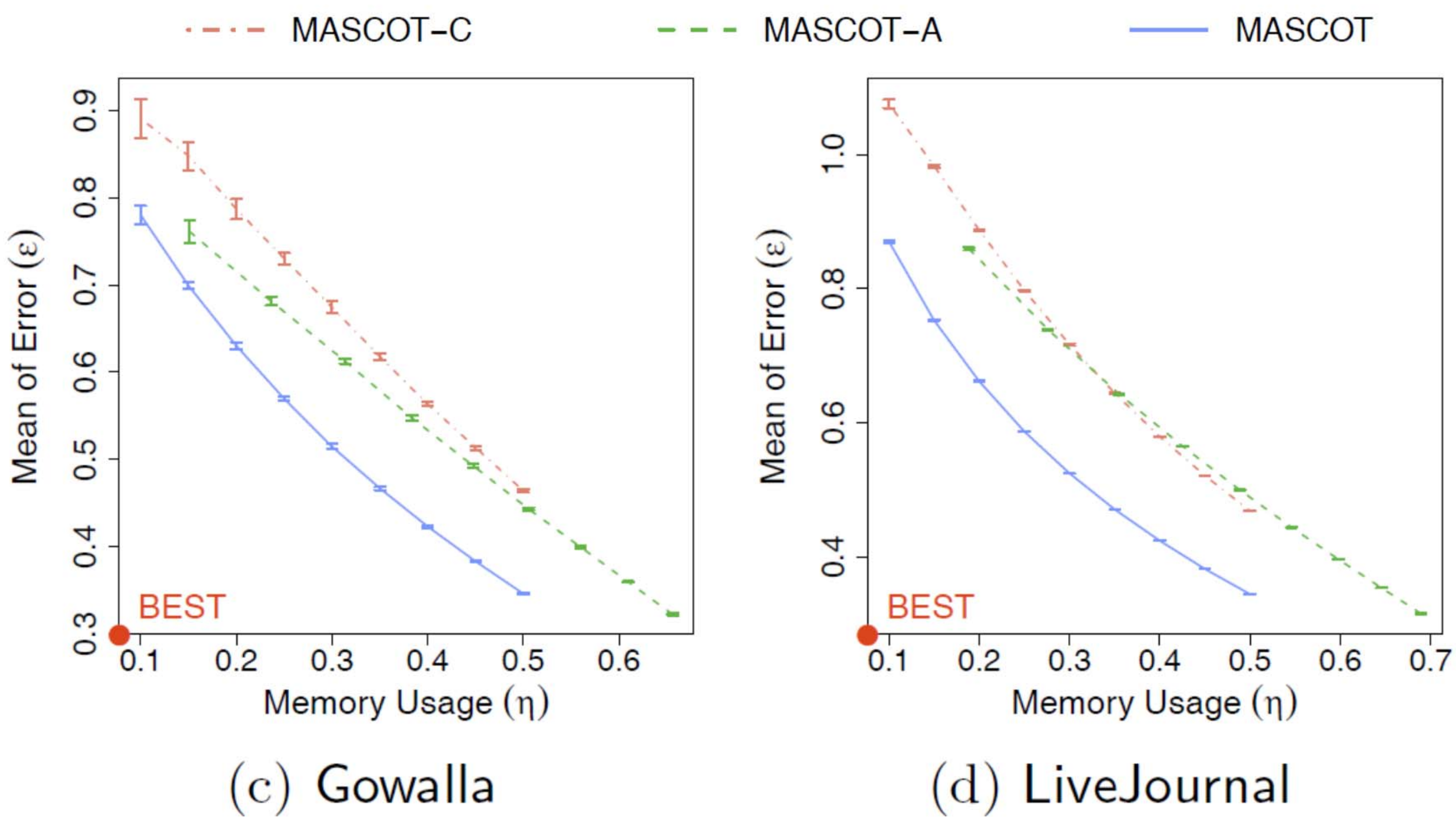
Memory Usage vs. Pearson Cor. Coeff.



(c) Gowalla (d) LiveJournal

[Lim et al., 2015 (KDD)]

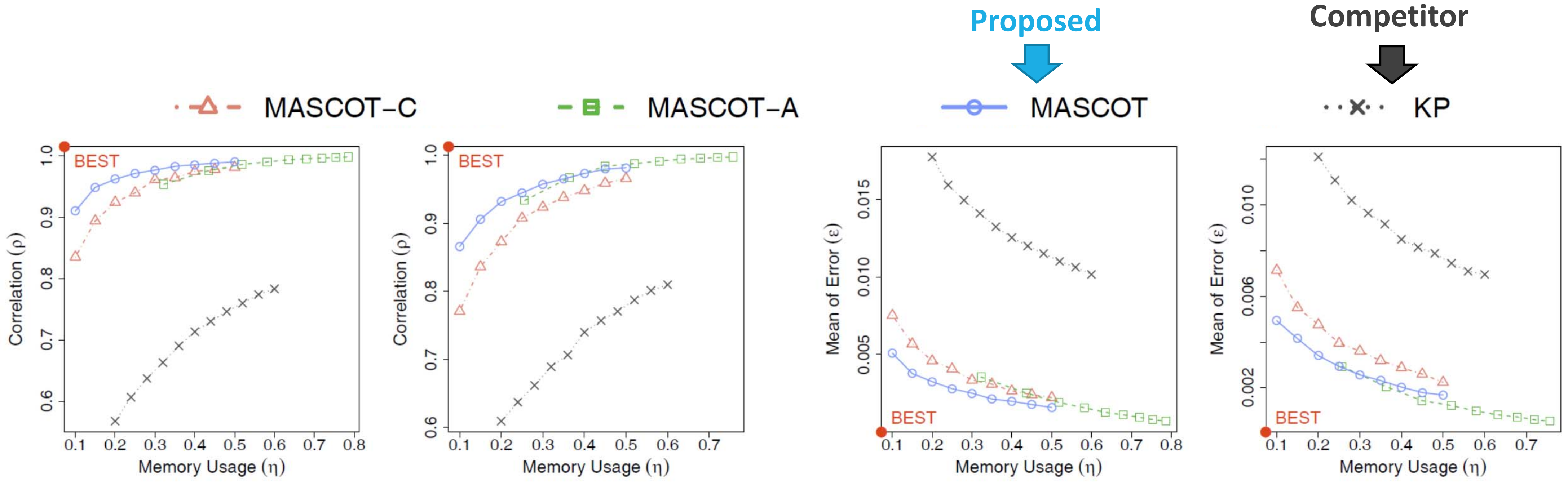
Memory Usage vs. Error



MASCOT outperforms two basic methods

[Lim et al., 2015 (KDD)]

Comparison to Existing Method



(a) Wiki-Conflict (b) Stanford (c) Wiki-Conflict (d) Stanford

Pearson Cor. Coeff. **Error**

Outline

Introduction

Finding Recently Frequent Items in Data Streams

Counting Local Triangles in Graph Streams

Conclusion

Conclusion

Large data streams are everywhere in recent days
- Memory-efficiency
- Real-time analysis

Method for finding top-k recently frequent items (TwMinSwap)
- CIKM 2014 (Yongsub Lim, Jihoon Choi, U Kang)

Method for counting local triangles in graph streams (MASCOT)
- KDD 2015 (Yongsub Lim, U Kang)

Improve existing methods and discover hidden patterns

References

Yongsub Lim, Jihoon Choi, and U Kang, Fast, Accurate, and Space-efficient Tracking of Time-weighted Frequent Items from Data Streams, 23rd ACM International Conference on Information and Knowledge Management (CIKM) 2014, Shaghai, China.

Yongsub Lim and U Kang, MASCOT: Memory-efficient and Accurate Sampling for Counting Local Triangles in Graph Streams, ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) 2015, Sydney, Australia

U Kang, Brendan Meeder, Evangelos E. Papalexakis, and Christos Faloutsos, Heigen: Spectral Analysis for Billion-Scale Graphs, IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 26, no.2, pp. 350-362, Feb. 2014.

Thank You !

http://kdmlab.org/