Efficient mining of complex networks

Luca Becchetti[1]

Università di Roma "La Sapienza" – Rome, Italy

1. Using what we learnt for graph mining

2. Web spam and Clustering
   - Web spam
     - Web spam Indices

1. Using what we learnt for graph mining

2. Web spam and Clustering
   - Web spam
     - Web spam Indices
   - Clustering

3. Computational challenges
   - Computational model

Using tools in practice ...

Data intensive graph mining

We have a data collection in the form of a large graph

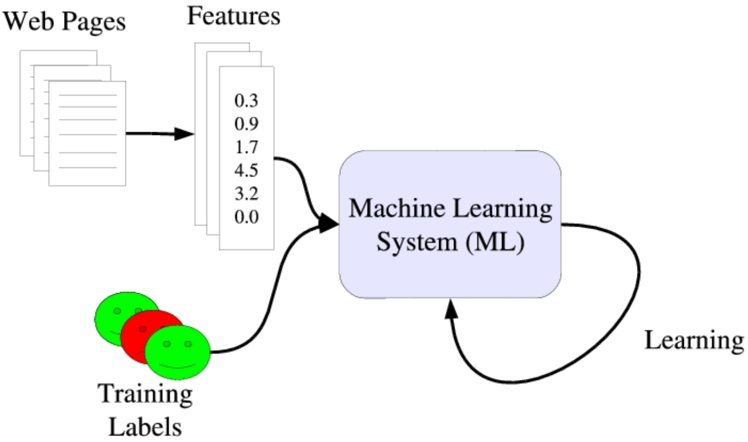
We have a mining task

- Document ranking
- Cyber-community detection
- Web spam detection
- Profiling of users accessing a search engine/on line store
  - Finding "typical" queries/items
  - Suggesting topics/items of potential interest to users who submitted/purchased a given query/item
- Detecting hot spots in epidemic spreading
- Topic distillation over hyperlinked document collections
- Detection of network bottlenecks

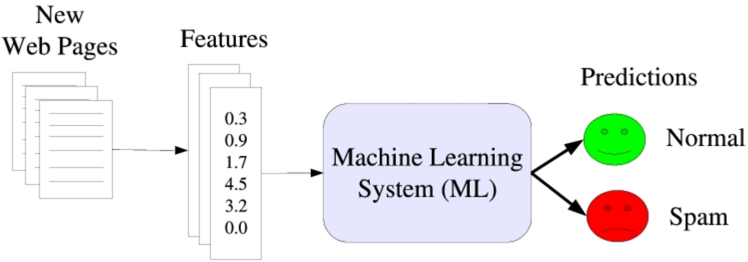
Mapping IR applications to indices

- Many data collections in the form of large scale graphs (e.g., Web crawls, query graphs)
- Many IR applications entail the computation of local indices on a per vertex basis
- Example: Pagerank ranking index
  - Requires a massive graph/matrix computation
  - Result is an index vector (Pagerank) with one component per Web page
- Different IR applications require computation of different indices

Automatic classifiers (e.g.: Web spam)



Automatic classifiers (cont.)



Challenges

Machine Learning Challenges:

- Learning with inter dependent variables (graph)

Information Retrieval Challenges:

Challenges

Machine Learning Challenges:

- Learning with inter dependent variables (graph)
- Learning with few examples

Information Retrieval Challenges:

Challenges

Machine Learning Challenges:

- Learning with inter dependent variables (graph)
- Learning with few examples
- Scalability

Information Retrieval Challenges:

Challenges

Machine Learning Challenges:

- Learning with inter dependent variables (graph)
- Learning with few examples
- Scalability

Information Retrieval Challenges:

- Feature extraction: which features?

Challenges

Machine Learning Challenges:

- Learning with inter dependent variables (graph)
- Learning with few examples
- Scalability

Information Retrieval Challenges:

- Feature extraction: which features?
- Feature aggregation: e.g., page/host/domain for the Web

Challenges

Machine Learning Challenges:

- Learning with inter dependent variables (graph)
- Learning with few examples
- Scalability

Information Retrieval Challenges:

- Feature extraction: which features?
- Feature aggregation: e.g., page/host/domain for the Web
- Recall/precision tradeoffs

Challenges

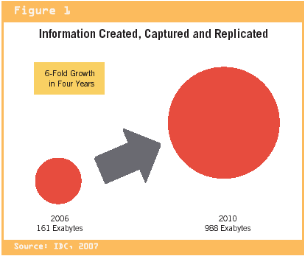
Machine Learning Challenges:

- Learning with inter dependent variables (graph)
- Learning with few examples
- Scalability

Information Retrieval Challenges:

- Feature extraction: which features?
- Feature aggregation: e.g., page/host/domain for the Web
- Recall/precision tradeoffs
- Scalability

Data size



- Indexable Web estimated to have more than 11.5 billion pages [Gulli and Signorini, 2005]
- As of now: roughly 100 times more (?)
- Facebook has about 1.5 billion users
- Amazon's unique monthly visitors: about 183 millions

General problem

- We are given a (typically large or huge) graph $G = (V, E)$
- Vertices may represent Web pages, people etc.
- Arcs (or edges) represent relationships. E.g., hyperlinks, email exchanges, social ties, interaction etc.
- Goal: compute, for every vertex, some index depending on the application and whose value depends on graph topology

Challenges

- Polynomial solutions may not suffice ...
- Graphs may be too large to fit in main memory
- Solutions must be scalable, both in memory and computational costs

This lecture

- Consider two exemplar applications
- See how techniques can be applied to these cases
  - Partial view, but gives flavour of techniques involved

Our motivating examples

- Web spam detection
  - Boost the Pagerank score of target Web pages
  - Uses content and/or link based techniques
  - We focus on link based spam
- Local clustering in massive graphs
  - Can unveil important aspects of the network's social structure (e.g., identify dense regions, communities etc.)

What is on the Web?

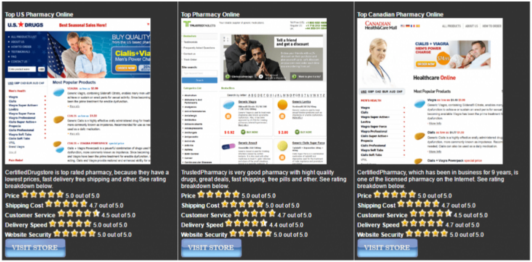
Information

What is on the Web?

Information + Porn

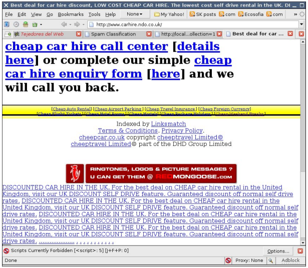
What is on the Web?

Information + Porn + On-line casinos + Free movies + Cheap software + Buy a MBA diploma + Prescription -free drugs + V!-4-gra + Get rich now now now!!!

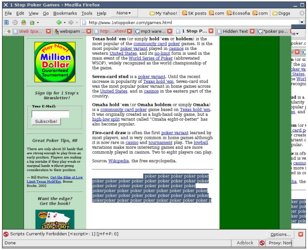


Forms of Web spamming

Typical Web Spam



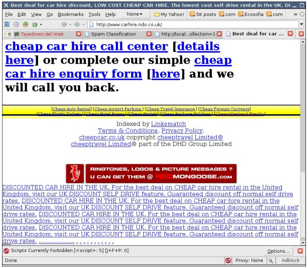
Hidden text



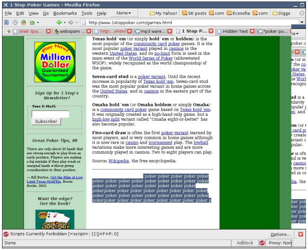
Many others...

Forms of Web spamming

Typical Web Spam



Hidden text

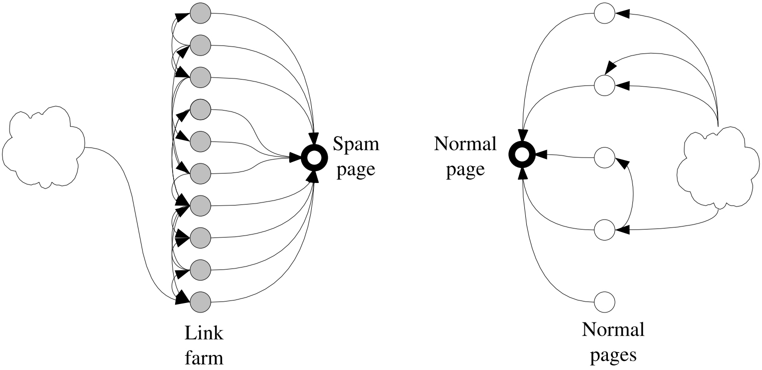


Many others...

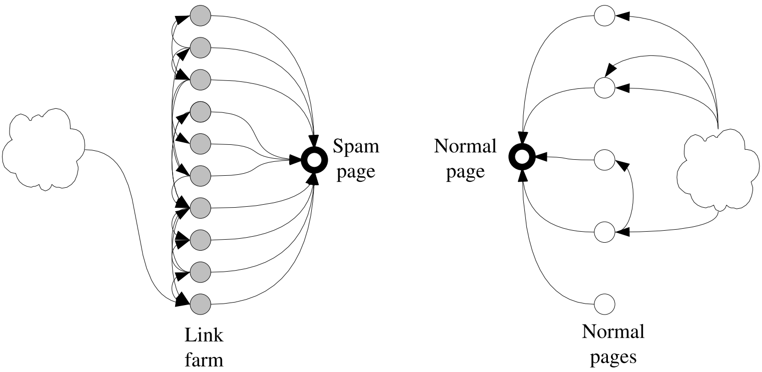
Adversarial relationship

Every undeserved gain in ranking for a spammer, is a loss of precision for the search engine.

Topological spam: link farms



Topological spam: link farms



Single-level farms can be detected by searching groups of nodes sharing their out-links [Gibson et al., 2005]

Motivation

[Fetterly et al., 2004] hypothesized that studying the distribution of statistics about pages could be a good way of detecting spam pages:

"**in a number of these distributions, outlier values are associated with web spam**"

Motivation

[Fetterly et al., 2004] hypothesized that studying the distribution of statistics about pages could be a good way of detecting spam pages:

"**in a number of these distributions, outlier values are associated with web spam**"

Research goal

Statistical analysis of link-based spam

Spam indices [Becchetti et al., 2007]

U.K. collection

18.5 million pages downloaded from the .UK domain in 2002

5,344 hosts manually classified (6% of the hosts)

Spam indices [Becchetti et al., 2007]

U.K. collection

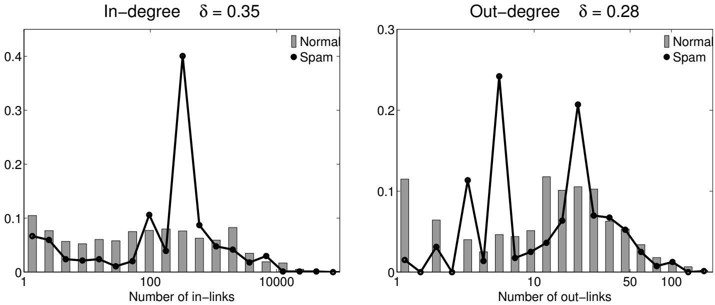
18.5 million pages downloaded from the .UK domain in 2002

5,344 hosts manually classified (6% of the hosts)

Classified entire hosts:

- ☑ A few hosts are mixed: spam and non-spam pages
- ☒ More coverage: sample covers 32% of the pages

Degree



($\delta =$ max. difference in C.D.F. plot)

PageRank

Let $\mathbf{P}_{N \times N}$ be the normalized adjacency matrix of a graph

- Row-normalized
- No “sinks”

Definition (PageRank)

Stationary state of:

$$\alpha \mathbf{P} + \frac{(1-\alpha)}{N} \mathbf{1}_{N \times N}$$

PageRank

Let $\mathbf{P}_{N \times N}$ be the normalized adjacency matrix of a graph

- Row-normalized
- No "sinks"

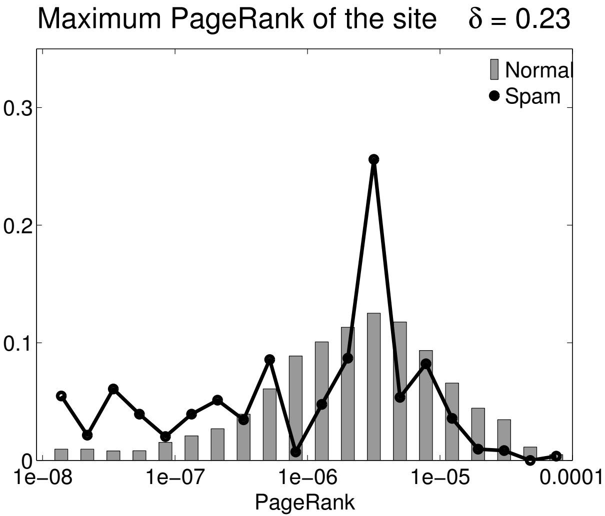
Definition (PageRank)

Stationary state of:

$$\alpha \mathbf{P} + \frac{(1-\alpha)}{N} \mathbf{1}_{N \times N}$$

- Follow links with probability α
  - Every link chosen with prob. $1/deg$.
- Random jump with probability $1-\alpha$

Maximum PageRank in the Host



TrustRank

TrustRank [Gyöngyi et al., 2004]

A node with high PageRank, but far away from a core set of "trusted nodes" is suspicious

TrustRank

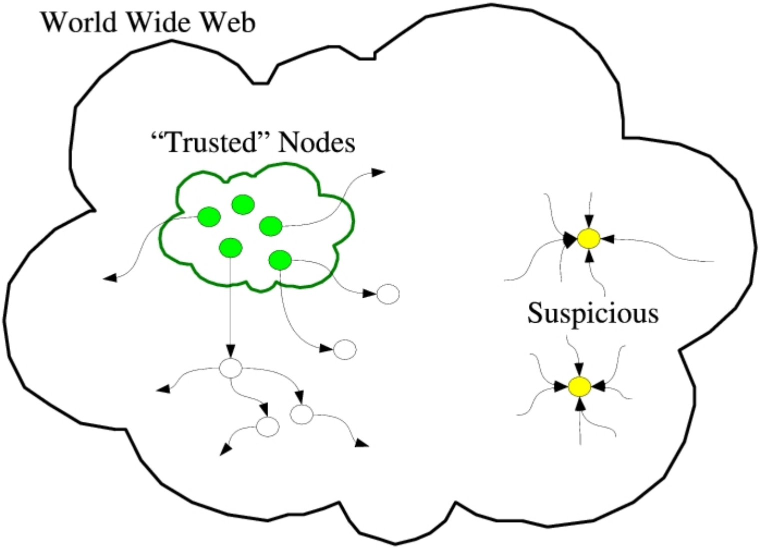
TrustRank [Gyöngyi et al., 2004]

A node with high PageRank, but far away from a core set of "trusted nodes" is suspicious

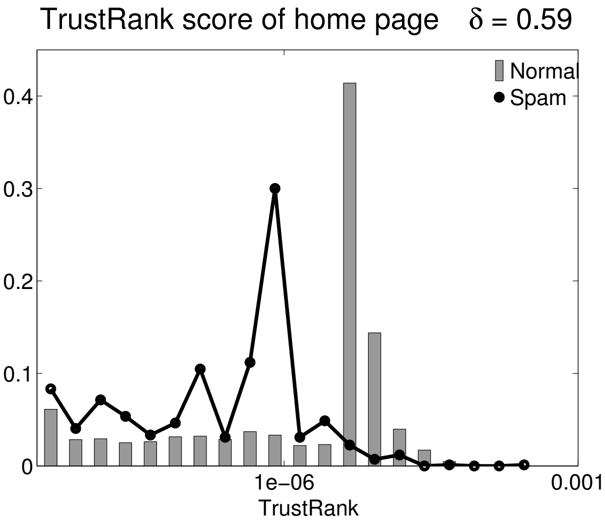
Start from a set of trusted nodes, then do a random walk, returning to the set of trusted nodes with probability $1 - \alpha$ at each step

ℹ Trusted nodes: data from `http://www.dmoz.org/`

TrustRank Idea

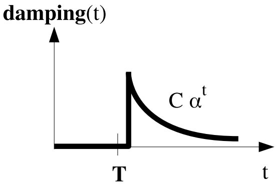


TrustRank score



Truncated PageRank

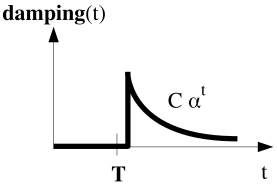
Reduce the direct contribution of the first levels of links:



$$\text{damping}(t) = \begin{cases} 0 & t \leq T \\ C\alpha^t & t > T \end{cases}$$

Truncated PageRank

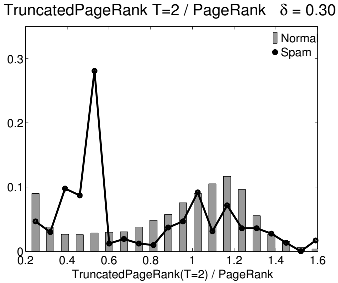
Reduce the direct contribution of the first levels of links:

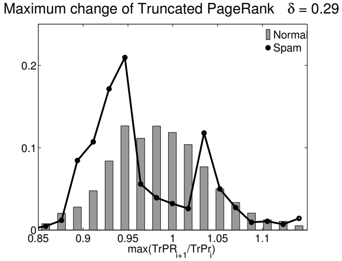


$$\text{damping}(t) = \begin{cases} 0 & t \le T \\ C\alpha^t & t > T \end{cases}$$

- ☑ No extra reading of the graph after PageRank
- ☑ Idea: most of spammers' rank due to pages that are few links away

Truncated PageRank(T=2) / PageRank

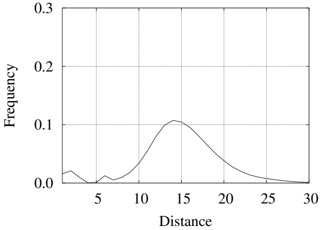




Idea: count "supporters" at different distances

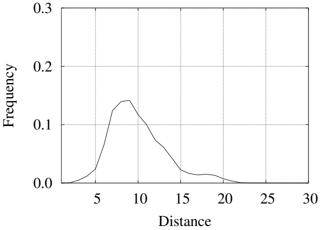
Number of *distinct* nodes at a given distance:

.UK 18 mill. nodes



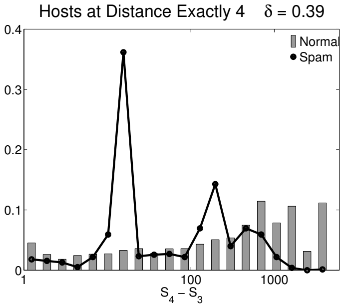
Average distance
14.9 clicks

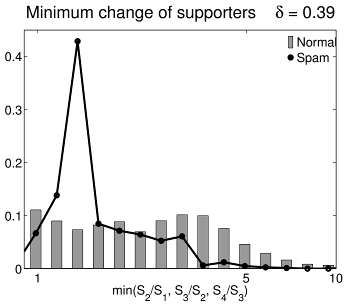
.EU.INT 860,000 nodes



Average distance
10.0 clicks

Supporters and their change





Clustering coefficient

- Compute triangle count for all vertices
- Local clustering coefficient and related statistics

Clustering coefficient

- Compute triangle count for all vertices
- Local clustering coefficient and related statistics

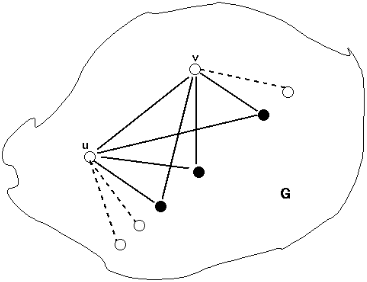
Motivation

- Analysis of social or biological networks [Newman, 2003]
- Thematic relationships in the Web [Eckmann and Moses, 2002]
- Common interests [Buchsbaum et al., 2003]

Web spam: [Fetterly et al., 2004] hypothesized that studying the distribution of statistics about pages could be a good way of detecting spam pages:

"**in a number of these distributions, outlier values are associated with web spam**"

Local Clustering Coefficient



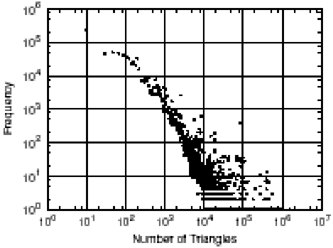
- $S(u) : \{v : (u, v) \in E\}$, $d(u) = |S(u)|$
- $T(u)$: No. triangles to which u belongs

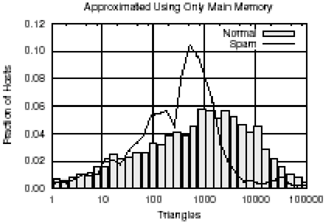
Clustering Coefficient

$CC_1 = \frac{2\sum_u T(u)}{\sum_u d(u)(d(u)-1)}$ (Alternative definition)

$\mathbf{CC_2} = \frac{\mathbf{1}}{|\mathbf{V}|} \sum_{\mathbf{u} \in \mathbf{V}} \frac{\mathbf{2T(u)}}{\mathbf{d(u)(d(u)-1)}}$

Distribution of triangles/clustering coefficient





- Distribution of number of triangles follows power law [Eckmann and Moses, 2002]
- Distributions of number of triangles/clustering coefficient in normal/spam pages
- Allows also to discriminate content quality in Yahoo! Answers [Becchetti et al., 2008]

Semi-streaming [Feigenbaum et al., 2004]

- Graph stored in secondary memory as adjacency or edge list
- **No random access possible**
- $O(N \log N)$ bits available in main memory
  - Limited amount of information per vertex
  - ✗ Not enough to store links in main memory
- Limited (constant or $O(\log N)$) number of passes
- ☑ No previous knowledge about graph
- Compute index for all vertices concurrently

More specifically:

We can store in main memory a (small) constant number of size N vectors with components of size $O(\log N)$ bits

Some previous work

- Computation of approximate matchings and distances [Feigenbaum et al., 2004, Feigenbaum et al., 2005]
- Lower bounds for neighbourhoods problems [Buchsbaum et al., 2003]
- Tradeoffs between number of passes and space for shortest path problems [Demetrescu et al., 2006]

Related: Streaming [Muthukrishnan, 2005]

- Stream of items accessed sequentially
- Maintain statistics (e.g., most frequent elements, histograms etc.)
- $O(\log \mathit{Space})$ overall, $O(\log \mathit{Time})$/item

General algorithm we consider

Require: N: number of nodes, d: distance, k: bits

```
for node : 1 ... N, bit: 1 ... k do
    INIT(node,bit)
end for
```

General algorithm we consider

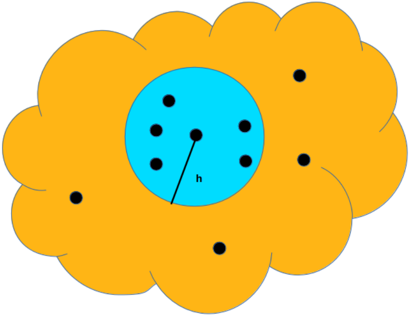
Require: N: number of nodes, d: distance, k: bits

for node : 1 ... N, bit: 1 ... k **do**
    INIT(node,bit)
end for
for distance : 1 ... d **do** {Iteration step}
    INIT(Aux)
    for src : 1 ... N **do** {Follow links in the graph}
        for all links from src to dest **do**
            Aux[src] $\leftarrow$ Combine(Aux[dest], V[src,$\cdot$])
        end for
    end for
    V $\leftarrow$ Aux
end for

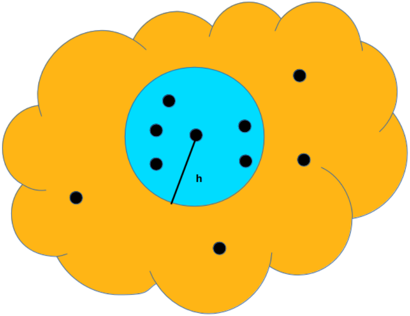
General algorithm we consider

```
Require: N: number of nodes, d: distance, k: bits
for node : 1 ... N, bit: 1 ... k do
    INIT(node,bit)
end for
for distance : 1...d do {Iteration step}
    INIT(Aux)
    for src : 1 ... N do {Follow links in the graph}
        for all links from src to dest do
            Aux[src] ← Combine(Aux[dest], V[src,·])
        end for
    end for
    V ← Aux
end for
for node: 1...N do {Estimation}
    Index[node] ← ESTIMATE( V[node,·] )
end for
return Index
```

Counting the number of supporters

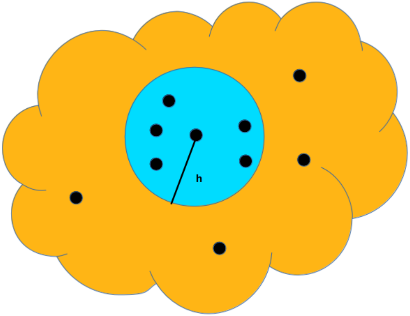


Counting the number of supporters



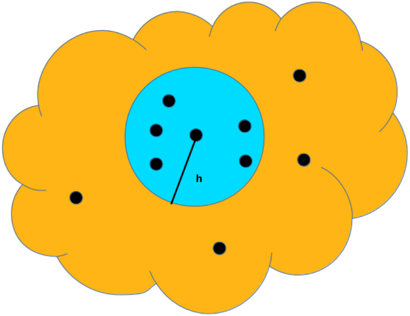
- For every node v, count the number of nodes within h hops

Counting the number of supporters



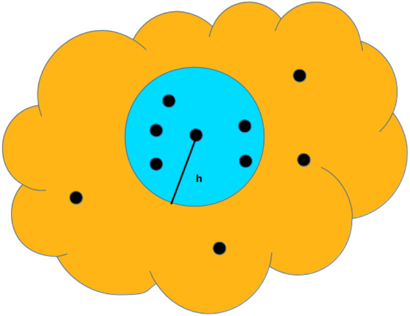
- For every node v, count the number of nodes within h hops
- Do this for different values of h

Counting the number of supporters



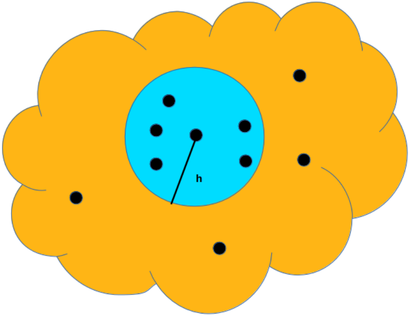
- For every node v, count the number of nodes within h hops
- Do this for different values of h
- Count each supporter *only once*

Counting the number of supporters



- For every node v, count the number of nodes within h hops
- Do this for different values of h
- Count each supporter *only once*
- Let $N(x, h) = \#$ nodes within h hops of x

Counting the number of supporters



- For every node v, count the number of nodes within h hops
- Do this for different values of h
- Count each supporter *only once*
- Let $N(x, h) = \#$ nodes within h hops of x
- *Can we directly apply the general algorithm seen before?*

A detour: back to distinct counting

Composing two sketches

- Assume two sets S_1 and S_2

A detour: back to distinct counting

Composing two sketches

- Assume two sets S_1 and S_2
- Assume $\mathbf{sk}(S_1) = (R_1(S_1), \ldots, R_k(S_1))$ and $\mathbf{sk}(S_2) = (R_1(S_2), \ldots, R_k(S_2))$ are corresponding sketches

A detour: back to distinct counting

Composing two sketches

- Assume two sets S_1 and S_2
- Assume $\mathbf{sk}(S_1) = (R_1(S_1), \ldots, R_k(S_1))$ and $\mathbf{sk}(S_2) = (R_1(S_2), \ldots, R_k(S_2))$ are corresponding sketches

What is $\mathbf{sk}(S_1 \cup S_2)$?

A detour: back to distinct counting

Composing two sketches

- Assume two sets S_1 and S_2
- Assume $\mathbf{sk}(S_1) = (R_1(S_1), \ldots, R_k(S_1))$ and $\mathbf{sk}(S_2) = (R_1(S_2), \ldots, R_k(S_2))$ are corresponding sketches

What is $\mathbf{sk}(S_1 \cup S_2)$?

Composability ...

$$\mathbf{sk}(S_1 \cup S_2) = (\max\{R_1(S_1), R_1(S_2)\}, \ldots, \max\{R_k(S_1), R_k(S_2)\})$$

Let $\text{Combine}(\mathbf{sk}(S_1), \mathbf{sk}(S_2)) = (\max\{R_1(S_1), R_1(S_2)\}, \ldots, \max\{R_k(S_1), R_k(S_2)\})$

Supporters: Probabilistic counting

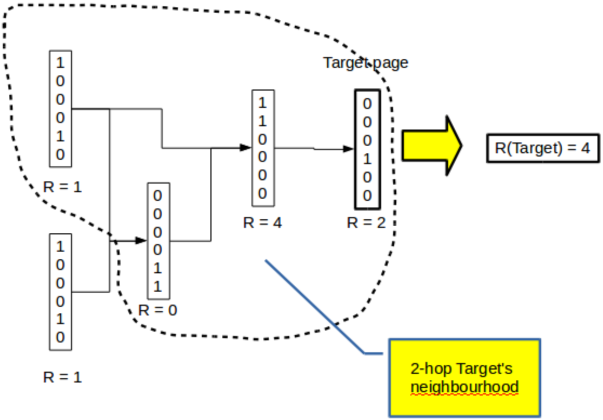
- Want to estimate $N(Target, 2)$

Supporters: Probabilistic counting

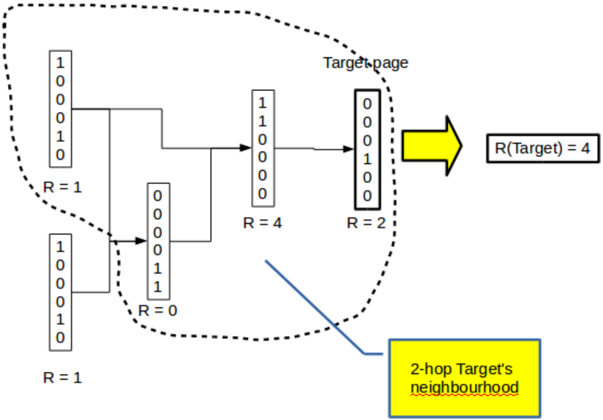
- Want to estimate $N(Target, 2)$
- View $N(Target, 2)$ as set

Supporters: Probabilistic counting

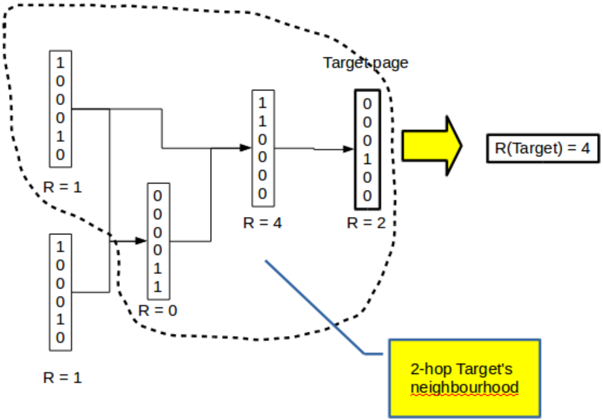
- Want to estimate $N(Target, 2)$
- View $N(Target, 2)$ as set
- Use distinct counting algorithm of [Alon et al., 1999]



Supporters: Probabilistic counting

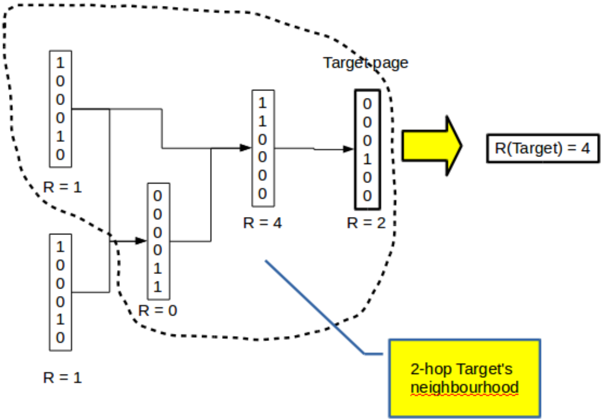


Supporters: Probabilistic counting



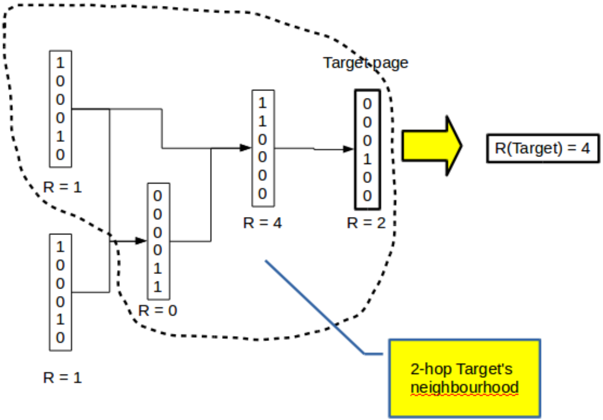
- Variant of ANF algorithm [Palmer et al., 2002] based on probabilistic counting [Flajolet and Martin, 1985]

Supporters: Probabilistic counting



- Variant of ANF algorithm [Palmer et al., 2002] based on probabilistic counting [Flajolet and Martin, 1985]
- Can be computed together with PageRank

Supporters: Probabilistic counting



- Variant of ANF algorithm [Palmer et al., 2002] based on probabilistic counting [Flajolet and Martin, 1985]
- Can be computed together with PageRank

ANF-like algorithm

Require: N: number of nodes, d: distance, k: bits

1: **for** node : 1 . . . N, bit: 1 . . . k **do**
2: INIT(node,bit) {Initialize node sketches}
3: **end for**

ANF-like algorithm

Require: N: number of nodes, d: distance, k: bits

for node : 1 ... N, bit: 1 ... k **do**
    INIT(node,bit) {Initialize node sketches}
end for
for distance : 1 ... d **do** {Iteration step}
    Aux $\leftarrow \mathbf{0}_k$
    for src : 1 ... N **do** {Follow links in the graph}
        for all links from src to dest **do**
            Aux[src] $\leftarrow$ Combine(Aux[dest], V[src,$\cdot$])
        end for
    end for
    V $\leftarrow$ Aux
end for

ANF-like algorithm

Require: N: number of nodes, d: distance, k: bits

```
for node : 1 ... N, bit: 1 ... k do
    INIT(node,bit) {Initialize node sketches}
end for
for distance : 1...d do {Iteration step}
    Aux ← 0_k
    for src : 1 ... N do {Follow links in the graph}
        for all links from src to dest do
            Aux[src] ← Combine(Aux[dest], V[src,·])
        end for
    end for
    V ← Aux
end for
for node: 1...N do {Estimate supporters}
    Supporters[node] ← ESTIMATE( V[node,·] )
end for
return Supporters
```

Our estimator

- The estimator proposed in [Alon et al., 1999]

Our estimator

- The estimator proposed in [Alon et al., 1999]
- For example, let $k = s \cdot t$ for suitable s and t

Our estimator

- The estimator proposed in [Alon et al., 1999]
- For example, let $k = s \cdot t$ for suitable s and t
- Let $(R_1, \ldots, R_t, \ldots, R_{(s-1)t+1}, \ldots, R_{st})$ be a generic node x neighbourhood's sketch

Our estimator

- The estimator proposed in [Alon et al., 1999]
- For example, let $k = s \cdot t$ for suitable s and t
- Let $(R_1, \ldots, R_t, \ldots, R_{(s-1)t+1}, \ldots, R_{st})$ be a generic node x neighbourhood's sketch
- For $i = 1, \ldots, s$: $\hat{R}_i = \frac{\sum_{j=1}^{t} R_{(i-1)t+j}}{t}$

Our estimator

- The estimator proposed in [Alon et al., 1999]
- For example, let $k = s \cdot t$ for suitable s and t
- Let $(R_1, \ldots, R_t, \ldots, R_{(s-1)t+1}, \ldots, R_{st})$ be a generic node x neighbourhood's sketch
- For $i = 1, \ldots, s$: $\hat{R}_i = \frac{\sum_{j=1}^{t} R_{(i-1)t+j}}{t}$
- $R(x) = median(\hat{R}_1, \ldots, \hat{R}_s)$

Our estimator

- The estimator proposed in [Alon et al., 1999]
- For example, let $k = s \cdot t$ for suitable s and t
- Let $(R_1, \ldots, R_t, \ldots, R_{(s-1)t+1}, \ldots, R_{st})$ be a generic node x neighbourhood's sketch
- For $i = 1, \ldots, s$: $\hat{R}_i = \frac{\sum_{j=1}^{t} R_{(i-1)t+j}}{t}$
- $R(x) = median(\hat{R}_1, \ldots, \hat{R}_s)$
- $\text{supporters}(x) = 2^{R(x)}$

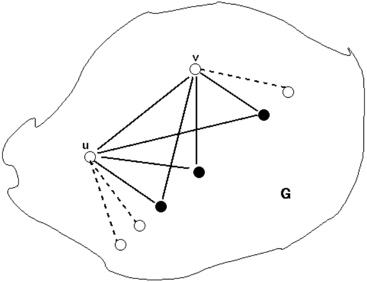
Our estimator

- The estimator proposed in [Alon et al., 1999]
- For example, let $k = s \cdot t$ for suitable s and t
- Let $(R_1, \ldots, R_t, \ldots, R_{(s-1)t+1}, \ldots, R_{st})$ be a generic node x neighbourhood's sketch
- For $i = 1, \ldots, s$: $\hat{R}_i = \frac{\sum_{j=1}^{t} R_{(i-1)t+j}}{t}$
- $R(x) = median(\hat{R}_1, \ldots, \hat{R}_s)$
- $\text{supporters}(x) = 2^{R(x)}$

Tuning

For a given value of k, s and t allow to trade off between accuracy and probability

Local Clustering Coefficient

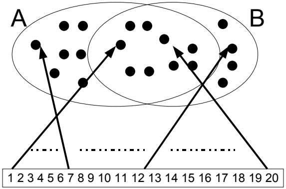


- $S(u) : \{v : (u, v) \in E\}$, $d(u) = |S(u)|$

Number of Triangles and Clustering Coefficient

- Estimate local clustering coefficient concurrently for all vertices
- Semi-streaming model
- Need to pass over the graph as few times as possible
- *Key step*: estimate size of neighbourhood intersection

Estimating Set Intersection: intuition



- Assume items of the universe initially numbered
- Any of the possible $n!$ permutations chosen u.a.r.
- Items reordered accordingly
- $\mathbf{P}[\min \pi(A) = \min \pi(B)] = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$

Estimating Set Intersection: basic technique

Approach assumes family of *minwise independent* permutations
[Broder, 1998, Broder, 2000, Broder et al., 1997]

Estimating Set Intersection: basic technique

Approach assumes family of *minwise independent* permutations [Broder, 1998, Broder, 2000, Broder et al., 1997]

In practice...

- Exponential space ($\Omega(n)$ bits) needed to represent minwise families [Broder et al., 1998]
- $\pi(x) = ((ax + b) \mod p) \mod n$, with a and b chosen u.a.r., p a large prime [Bohman et al., 2000]

Triangles: Ideal algorithm

If we new $J(S(u), S(v))$:

$$T_{uv} = |S(u) \cap S(v)| = \frac{J}{J+1}(|S(u)| + |S(v)|)$$

- m independent trials
- Z_{uv}: # times that $\min \pi(S(u)) = \min \pi(S(v))$

Our estimator:

$$\overline{T}_{uv} = \frac{Z_{uv}}{Z_{uv} + m}(|S(u)| + |S(v)|)$$

We use a more efficient modified alg in practice

High probability bound

$$\mathbf{P}\left[|\overline{T}_{uv} - T_{uv}| > \epsilon T_{uv}\right] \leq$$
$$\leq Ce^{-\frac{\epsilon^2}{3} mJ(S(u),S(v))}.$$

for a suitable constant C

General Algorithm

$\mathbf{Z} = \mathbf{0}$
for i: 1 ... m **do** {Independent trials}
  for u : 1 ... $|V|$ **do** {Assign labels}
    $l_i(u) = \text{hash}_i(u)$ {Minwise linear permutation}
  end for

General Algorithm

```
Z = 0
for i: 1 ... m do {Independent trials}
    for u : 1 ... |V| do {Assign labels}
        l_i(u) = hash_i(u) {Minwise linear permutation}
    end for
    for u : 1 ... |V| do {Compute fingerprints}
        F_i(u) = min_{v∈S(u)} l_i(u)
    end for{1 scan of G}
```

General Algorithm

$\mathbf{Z} = \mathbf{0}$
for i: 1 ... m **do** {Independent trials}
    for u : 1 ... $|V|$ **do** {Assign labels}
        $l_i(u) = \text{hash}_i(u)$ {Minwise linear permutation}
    end for
    for u : 1 ... $|V|$ **do** {Compute fingerprints}
        $F_i(u) = \min_{v \in S(u)} l_i(u)$
    end for{1 scan of G}
    for u : 1 ... $|V|$ **do** {Update counters}
        for $v \in S(u)$ **do**
            if $F_i(u) == F_i(v)$ **then** {Minima are equal}
                $Z_{uv} = Z_{uv} + 1$ {Z_{uv}'s stored on disk}
            end if
        end for
    end for
end for

Estimating Triangles/cont.

- $\overline{T}_{uv} = \frac{Z_{uv}}{Z_{uv}+m}(d(u)+d(v))$ is our estimate of $|S(u) \cap S(v)|$
- $\overline{T}(u) = \frac{1}{2}\sum_{v \in S(v)} \overline{T}_{uv}$ is our estimate of $T(u)$
- In practice, $m = O(\log N)$

Implementation

- The Z_{uv}'s must be stored on disk (size of $\mathbf{Z}$ same order as adjacency list)

- *For every i, updating Z_{uv} requires access to disk*
- *Computing counters most expensive operation*

Using the adjacency matrix ([Tsourakakis, 2008])

Let $\mathbf{A}$ denote the adjacency matrix of an *undirected* graph

- Consider $\mathbf{A}^3$

Using the adjacency matrix ([Tsourakakis, 2008])

Let $\mathbf{A}$ denote the adjacency matrix of an *undirected* graph

- Consider $\mathbf{A}^3$
- $\mathbf{A}^3_{ii} = 2$ (# triangles incident in i)

Using the adjacency matrix ([Tsourakakis, 2008])

Let $\mathbf{A}$ denote the adjacency matrix of an *undirected* graph

- Consider $\mathbf{A}^3$
- $\mathbf{A}^3_{ii} = 2$ ($\#$ triangles incident in i)
- Each triangle counted twice

Using the adjacency matrix ([Tsourakakis, 2008])

Let $\mathbf{A}$ denote the adjacency matrix of an *undirected* graph

- Consider $\mathbf{A}^3$
- $\mathbf{A}^3_{ii} = 2$ (# triangles incident in i)
- Each triangle counted twice

As a consequence...

$Trace(\mathbf{A^3}) = 6$ (# triangles)
Reason: triangle (i, j, k) contributes twice to $\mathbf{A}^3_{ii}$, $\mathbf{A}^3_{jj}$ and $\mathbf{A}^3_{kk}$

Spectra and triangles

Recall that **A** is symmetric ...
hence it can be diagonalized:

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{u_i} \mathbf{u_i}^T,$$

where $(\lambda_i, \mathbf{u_i})$ is the i-th eigenpair

As a consequence...

$$\mathbf{A}^3 = \sum_{i=1}^{n} \lambda_i^3 \mathbf{u_i} \mathbf{u_i}^T$$

Theorem ([Tsourakakis, 2008])

Let $\Delta(G) = \#$ triangles and $\mathbf{A}$ the adjacency matrix of G.
Let $\Delta_i(G) = \#$ triangles in which i is involved. We have:

$$\Delta(G) = \frac{1}{6}\sum_{i=1}^{n}\lambda_i^3$$

$$\Delta_i(G) = \frac{1}{2}\sum_{j=1}^{n}\lambda_i^3\mathbf{u_j}(\mathbf{i})^2,$$

with $\mathbf{u_i}(j)$ the j-th component of $\mathbf{u_i}$

Proof sketch

- First claim follows since trace of a matrix $= \sum$ eigenvalues
- Second claim follows from expression of $\mathbf{A}_{ii}^3$ in spectral decomposition

In practice ...

Graphs we are interested in normally obey power laws
Same applies to distribution of triangles

Implications

- Most triangles incident to relatively small fraction of nodes
- Enough to sum over the first k entries of $\mathbf{A}^3$'s diagonal - k relatively small
- Corresponds to computing the first k eigenvectors of $\mathbf{A}$

Alon, N., Matias, Y., and Szegedy, M. (1999).
The space complexity of approximating the frequency moments.
Journal of Computer and System Sciences, 58(1):137–147.

Becchetti, L., Boldi, P., Castillo, C., and Gionis, A. (2008).
Efficient semi-streaming algorithms for local triangle counting in massive graphs.
In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2008)*, pages 16–24. ACM Press.

Becchetti, L., Castillo, C., Donato, D., Leonardi, S., and Baeza-Yates, R. (2006a).
Link-based characterization and detection of Web Spam.
In *Second International Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, Seattle, USA.

Becchetti, L., Castillo, C., Donato, D., Leonardi, S., and Baeza-Yates, R. (2006b).
Using rank propagation and probabilistic counting for link-based spam detection.
In *Proceedings of the Workshop on Web Mining and Web Usage Analysis (WebKDD)*, Pennsylvania, USA. ACM Press.

Becchetti, L., Castillo, C., Donato, D., Leonardi, S., and Baeza-Yates, R. (2007).
Link analysis for web spam detection.
ACM Trans. on the Web, 2(1):1–42.

Bohman, T., Cooper, C., and Frieze, A. M. (2000).
Min-wise independent linear permutations.
Electr. J. Comb, 7.

Broder, A. Z. (1998).
On the resemblance and containment of documents.
In *Compression and Complexity of Sequences, IEEE Computer Society*.

Broder, A. Z. (2000).

Identifying and filtering near-duplicate documents.

In *Combinatorial Pattern Matching, 11th Annual Symposium, CPM 2000, Montreal, Canada, June 21-23, 2000, Proceedings*, volume 1848 of *Lecture Notes in Computer Science*, pages 1–10. Springer.

Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. (1998).

Min-wise independent permutations (extended abstract).

In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 327–336, New York, NY, USA. ACM Press.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997).

Syntactic clustering of the web.

In *Selected papers from the sixth international conference on World Wide Web*, pages 1157–1166, Essex, UK. Elsevier Science Publishers Ltd.

Buchsbaum, A. L., Giancarlo, R., and Westbrook, J. (2003).
On finding common neighborhoods in massive graphs.
Theor. Comput. Sci, 1-3(299):707–718.

Demetrescu, C., Finocchi, I., and Ribichini, A. (2006).
Trading off space for passes in graph streaming problems.
In *Proceedings of the 7th annual ACM-SIAM Symposium on Discrete Algorithms*.

Eckmann, J.-P. and Moses, E. (2002).
Curvature of co-links uncovers hidden thematic layers in the world wide web.
PNAS, 99(9):5825–5829.

Feigenbaum, J., Kannan, S., Gregor, M. A., Suri, S., and Zhang, J. (2004).
On graph problems in a semi-streaming model.
In *31st International Colloquium on Automata, Languages and Programming*.

Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., and Zhang, J. (2005).
Graph distances in the streaming model: the value of space.
In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 745–754. SIAM.

Fetterly, D., Manasse, M., and Najork, M. (2004).
Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages.
In *Proceedings of the seventh workshop on the Web and databases (WebDB)*, pages 1–6, Paris, France.

Flajolet, P. and Martin, N. G. (1985).
Probabilistic counting algorithms for data base applications.
Journal of Computer and System Sciences, 31(2):182–209.

Gibson, D., Kumar, R., and Tomkins, A. (2005).
Discovering large dense subgraphs in massive graphs.
In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 721–732. VLDB Endowment.

Gulli, A. and Signorini, A. (2005).
The indexable Web is more than 11.5 billion pages.
In *Poster proceedings of the 14th international conference on World Wide Web*, pages 902–903, Chiba, Japan. ACM Press.

Gyöngyi, Z., Garcia-Molina, H., and Pedersen, J. (2004).
Combating Web spam with TrustRank.
In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB)*, pages 576–587, Toronto, Canada. Morgan Kaufmann.

Muthukrishnan (2005).
Data streams: Algorithms and applications.
In *Foundations and Trends in Theoretical Computer Science, Now Publishers or World Scientific*, volume 1.

Newman, M. E. J. (2003).
The structure and function of complex networks.
SIAM Review, 45(2):167–256.

Palmer, C. R., Gibbons, P. B., and Faloutsos, C. (2002).
ANF: a fast and scalable tool for data mining in massive graphs.
In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 81–90, New York, NY, USA. ACM Press.

Tsourakakis, C. E. (2008).
Fast counting of triangles in large real networks without counting: Algorithms and laws.
In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 608–617. IEEE.