Fast Asynchronous Anti-TrustRank for Web Spam Detection

Joyce Jiyoung Whang*
Sungkyunkwan University (SKKU)
Suwon, Korea
jjwhang@skku.edu

Yeon Seong Jeong
Sungkyunkwan University (SKKU)
Suwon, Korea
jys3548@skku.edu

Inderjit S. Dhillon
University of Texas at Austin
Austin, TX, USA
inderjit@cs.utexas.edu

Seonggoo Kang
Naver Corporation
Seongnam, Korea
seong.goo.kang@navercorp.com

Jungmin Lee
Naver Corporation
Seongnam, Korea
jmin.lee@navercorp.com

ABSTRACT

Web spam detection is an important problem in Web search. Since Web spam pages tend to have a lot of spurious links, many Web spam detection algorithms exploit the hyperlink structure between the Web pages to detect the spam pages. Anti-TrustRank algorithm is a well-known link-based spam detection algorithm which follows the principle that spam pages are likely to be referenced by other spam pages. Since a real-world Web graph involves tens of billions of nodes, it is crucial to develop work-efficient Web spam detection algorithms. In this paper, we develop asynchronous Anti-TrustRank algorithms which allow us to significantly reduce the number of arithmetic operations compared to the traditional synchronous Anti-TrustRank algorithm without degrading the performance in detecting Web spams. We theoretically prove the convergence of the asynchronous Anti-TrustRank algorithms, and conduct experiments on a real-world Web graph indexed by NAVER which is the most popular search engine in Korea.

ACM Reference Format:
Joyce Jiyoung Whang, Yeon Seong Jeong, Inderjit S. Dhillon, Seonggoo Kang, and Jungmin Lee. 2018. Fast Asynchronous Anti-TrustRank for Web Spam Detection. In *Proceedings of WSDM workshop on Misinformation and Misbehavior Mining on the Web (MIS2)*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Web spam detection is one of the most important tasks in Web search. Given a Web graph with a set of nodes and edges where a node indicates a Web document and an edge indicates a hyperlink between documents, search engines rank the documents based on the link structure, e.g., [3], [9], and [11]. Web spams refer to the Web documents that have a lot of spurious links (e.g., creating link farms [21]) to mislead the search engines [7]. Therefore, it is critical for a search engine to correctly detect the Web spams to provide reliable search results.

A number of Web spam detection methods have been proposed over the years including link-based spam detection and content-based spam detection [17]. In particular, link analysis has been considered to be an important feature of a good spam detection method [1]. For example, [6] measures the impact of link spamming on a page's rank, and [4] provides a way to detect nepotistic links.

TrustRank [7] is one of the well-known link-based spam detection methods. The main idea of the TrustRank method is that good pages tend to point to other good pages, which leads to computing TrustRank scores of the nodes. Similarly, the Anti-TrustRank algorithm [10] has also been proposed with an intuition that spam pages tend to be pointed by other spam pages. It has been shown that Anti-TrustRank is able to achieve a higher precision than the TrustRank method. A method of propagating both trust and distrust values also has been considered [22]. Indeed, computing the TrustRank or Anti-TrustRank scores can be interpreted as computing a personalized PageRank [8] (also called as a biased PageRank) on the Web graph with a set of carefully selected seeds. A number of variations of PageRank have been proposed to accelerate PageRank computations including [2], [13], [23], and [5]. In particular, [20] has proposed a multi-threaded data-driven PageRank algorithm.

Since a Web graph usually involves tens of billions of nodes, it is crucial to develop a work-efficient algorithm for Web spam detection. In this paper, we design asynchronous Anti-TrustRank algorithms which are able to significantly reduce the number of arithmetic operations compared to the traditional synchronous Anti-TrustRank method while achieving the same accuracy. We theoretically prove the convergence of the asynchronous Anti-TrustRank methods. On a real-world Web graph provided by NAVER corporation which is the largest search engine company in Korea, we empirically observe that our residual-based asynchronous Anti-TrustRank algorithm allows us to compute the Anti-TrustRank scores with only 10% of the arithmetic operations required for the synchronous Anti-TrustRank method without degrading the performance in detecting Web spams.

2 ANTI-TRUSTRANK ALGORITHM

The principle behind the Anti-TrustRank [10] algorithm is that spam pages are likely to be referred by other spam pages. Given a graph $G = (\mathcal{V}, \mathcal{E})$, the Anti-TrustRank method first selects a set of seeds which consists of manually examined spam documents. An Anti-TrustRank (ATR) score is assigned to each document such that a document with a high ATR score is considered as a spam

*Corresponding author.

MIS2, 2018, Marina Del Rey, CA, USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

Algorithm 1: Synchronous ATR

Input: $G' = (\mathcal{V}, \mathcal{E}')$, $\mathcal{S}$, α, ϵ
Output: ATR vector $\mathbf{x}$
Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}_s$
while true **do**
  for $i \in \mathcal{V}$ **do**
    if $i \in \mathcal{S}$ **then**
      $x_i^{new} = \alpha \sum_{j \in Q_i} \frac{x_j}{|\mathcal{T}_j|} + (1-\alpha)$
    else
      $x_i^{new} = \alpha \sum_{j \in Q_i} \frac{x_j}{|\mathcal{T}_j|}$
    end if
    $\delta_i = |x_i^{new} - x_i|$
  end for
  $\mathbf{x} = \mathbf{x}^{new}$
  if $\|\delta\|_\infty < \epsilon$ **then**
    break;
  end if
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

Algorithm 2: Asynchronous ATR

Input: $G' = (\mathcal{V}, \mathcal{E}')$, $\mathcal{S}$, α, ϵ
Output: ATR vector $\mathbf{x}$
Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}_s$
for $i \in \mathcal{V}$ **do**
  `worklist.push(i)`
end for
while `!worklist.isEmpty` **do**
  i = `worklist.pop()`
  if $i \in \mathcal{S}$ **then**
    $x_i^{new} = \alpha \sum_{j \in Q_i} \frac{x_j}{|\mathcal{T}_j|} + (1-\alpha)$
  else
    $x_i^{new} = \alpha \sum_{j \in Q_i} \frac{x_j}{|\mathcal{T}_j|}$
  end if
  if $|x_i^{new} - x_i| \geq \epsilon$ **then**
    $x_i = x_i^{new}$
    for $j \in \mathcal{T}_i$ **do**
      if j is not in `worklist` **then**
        `worklist.push(j)`
      end if
    end for
  end if
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

Algorithm 3: Residual-based Asynchronous ATR

Input: $G' = (\mathcal{V}, \mathcal{E}')$, $\mathcal{S}$, α, ϵ
Output: ATR vector $\mathbf{x}$
Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}_s$
Initialize $\mathbf{r} = (1-\alpha)\alpha \boldsymbol{P}^T \mathbf{e}_s$
for $i \in \mathcal{V}$ **do**
  `worklist.push(i)`
end for
while `!worklist.isEmpty` **do**
  i = `worklist.pop()`
  $x_i^{new} = x_i + r_i$
  for $j \in \mathcal{T}_i$ **do**
    $r_j^{old} = r_j$
    $r_j = r_j + \frac{r_i \alpha}{|\mathcal{T}_i|}$
    if $r_j \geq \epsilon$ and $r_j^{old} < \epsilon$ **then**
      `worklist.push(j)`
    end if
  end for
  $r_i = 0$
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

document. The ATR scores of the seed spam documents are initialized to be one whereas the ATR scores of the rest of the nodes are initialized to be zero. From the seeds, the ATR scores are propagated to incoming neighbors of the nodes so that the documents having links to the spam documents end up with having high ATR scores.

2.1 Selecting Seeds

It is important to select good seeds in the Anti-TrustRank algorithm since the ATR scores are propagated from the selected seeds. One way to select good seed nodes is to consider PageRank scores [7], [10] because nodes with high PageRank scores are likely to be highly ranked by search engines, and it is critical to filter out spam documents which otherwise can be potentially exposed to users. Thus, we compute PageRank scores of the nodes, and select top-ranked nodes. Let $\mathcal{L}$ denote the set of selected nodes. Then, human experts classify the nodes in $\mathcal{L}$ into two classes: spam documents or normal documents. Let $\mathcal{S}$ denote the set of spam documents among the nodes in $\mathcal{L}$. Note that $\mathcal{S}$ is a subset of $\mathcal{L}$.

2.2 Synchronous Anti-TrustRank

The mechanism of how to compute the ATR scores is very similar to that of the personalized PageRank computation [8] [14]. The difference is that the ATR scores are propagated backward along with incoming links. Let $G' = (\mathcal{V}, \mathcal{E}')$ denote a graph with reverse edges, i.e., if an edge $\{i, j\} \in \mathcal{E}$ then $\{j, i\} \in \mathcal{E}'$. Also, let $\boldsymbol{A}$ denote the adjacency matrix of G'. Then, computing ATR is identical to computing the personalized PageRank on $\boldsymbol{A}$ with the personalized vector such that the positions of the seed spam documents (i.e., the nodes in $\mathcal{S}$) have ones and other values are zeros. Let Q_i denote the set of incoming neighbors of node i on G', and $\mathcal{T}_i$ denote the set of outgoing neighbors of node i on G'. Let $\mathbf{x}$ denote a vector of the ATR scores, and $\mathbf{e}_s$ denote a vector with ones for the positions of the seed spam documents and zeros for other positions. Also, let α denote the damping factor (we use $\alpha = 0.85$ throughout the paper), and ϵ denote the tolerance. We assume that there is no self-loop in the graph, i.e., the diagonal elements of $\boldsymbol{A}$ are all zeros. Algorithm 1 is a synchronous Anti-TrustRank algorithm where the ATR scores are updated only after all the nodes re-compute the ATR scores.

3 ASYNCHRONOUS ANTI-TRUSTRANK

We design asynchronous Anti-TrustRank algorithms by considering the Gauss-Seidel method. Instead of updating the ATR scores of all the nodes at every iteration, we maintain a `worklist` which contains a set of nodes whose ATR scores need to be updated. Initially, the `worklist` contains the entire vertices, and whenever we process a node from the `worklist`, we add the outgoing neighbors of the processed node (on G') to the `worklist`. Indeed, for the global PageRank problem [3], a scalable data-driven PageRank algorithm [20] has been considered in a multi-threaded programming environment [16]. We extend this idea to the ATR computation which is shown in Algorithm 2.

3.1 Convergence of Asynchronous ATR

By extending the analysis of [15], we show the convergence of Algorithm 2.

Theorem 1. *In Algorithm 2, when $x_i^{(k)}$ is updated to $x_i^{(k+1)}$, the total residual is decreased at least by $r_i(1-\alpha)$.*

Proof. The ATR vector $\mathbf{x}$ is computed as follows:

$$\mathbf{x} = \alpha \boldsymbol{P}^T \mathbf{x} + (1-\alpha)\mathbf{e}_s$$

where $\boldsymbol{P}$ is defined as $\boldsymbol{P} \equiv \boldsymbol{D}^{-1}\boldsymbol{A}$ ($\boldsymbol{D}$ is the degree diagonal matrix) and $\mathbf{e}_s$ is the personalized vector. This is the linear system of

$$(\boldsymbol{I} - \alpha \boldsymbol{P}^T)\mathbf{x} = (1-\alpha)\mathbf{e}_s$$

and the residual is defined to be

$$\mathbf{r} = (1-\alpha)\mathbf{e}_s - (\boldsymbol{I} - \alpha \boldsymbol{P}^T)\mathbf{x} = \alpha \boldsymbol{P}^T \mathbf{x} + (1-\alpha)\mathbf{e}_s - \mathbf{x}.$$

Let $x_i^{(k)}$ denote the k-th update of x_i. Since we initialize $\mathbf{x}$ as $\mathbf{x} = (1-\alpha)\mathbf{e}_s$, the initial residual $\mathbf{r}^{(0)}$ can be written as follows:

$$\mathbf{r}^{(0)} = (1-\alpha)\mathbf{e}_s - (\boldsymbol{I} - \alpha \boldsymbol{P}^T)(1-\alpha)\mathbf{e}_s = (1-\alpha)\alpha \boldsymbol{P}^T \mathbf{e}_s \geq 0. \quad (1)$$

For each node i from the `worklist`, we update its ATR value as follows:

[Case 1] $i \in \mathcal{S}$

$$x_i^{(k+1)} = (1-\alpha) + \alpha \sum_{j \in Q_i} \frac{x_j^{(k)}}{|\mathcal{T}_j|},$$

$$x_i^{(k+1)} = x_i^{(k)} + \underbrace{(1-\alpha) - x_i^{(k)} + \alpha[P^T \mathbf{x}^{(k)}]_i}_{r_i^{(k)}} = x_i^{(k)} + r_i^{(k)}.$$

[Case 2] $i \notin \mathcal{S}$

$$x_i^{(k+1)} = \alpha \sum_{j \in Q_i} \frac{x_j^{(k)}}{|\mathcal{T}_j|},$$

$$x_i^{(k+1)} = x_i^{(k)} \underbrace{- x_i^{(k)} + \alpha[P^T \mathbf{x}^{(k)}]_i}_{r_i^{(k)}} = x_i^{(k)} + r_i^{(k)}.$$

Thus, we see that

$$x_i^{(k+1)} = x_i^{(k)} + r_i^{(k)}. \tag{2}$$

Also, after such an update, we can show that $\mathbf{r}^{(k+1)} \geq 0$. Let $\gamma = \mathbf{r}_i^{(k)}$.

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \gamma \mathbf{e}_i \\ \mathbf{r}^{(k+1)} &= (1-\alpha)\mathbf{e}_s - (I - \alpha P^T)\mathbf{x}^{(k+1)} \\ \mathbf{r}^{(k+1)} &= (1-\alpha)\mathbf{e}_s - (I - \alpha P^T)(\mathbf{x}^{(k)} + \gamma \mathbf{e}_i) \\ \mathbf{r}^{(k+1)} &= \mathbf{r}^{(k)} - \gamma(I - \alpha P^T)\mathbf{e}_i \end{aligned} \tag{3}$$

Note that the i-th component of $\mathbf{r}^{(k+1)}$ goes to zero, and we only add positive values to the other components. Since the initial residual is positive shown in (1), we can see that $\mathbf{r}^{(k+1)} \geq 0$.

Now, by multiplying $\mathbf{e}^T$ in (3), we get:

$$\mathbf{e}^T \mathbf{r}^{(k+1)} = \begin{cases} \mathbf{e}^T \mathbf{r}^{(k)} - r_i^{(k)}(1-\alpha) & : \mathcal{T}_i \neq \emptyset \\ \mathbf{e}^T \mathbf{r}^{(k)} - r_i^{(k)} & : \mathcal{T}_i = \emptyset \end{cases}$$

This implies that when a node i's ATR value is updated, its residual r_i becomes zero, and $\alpha r_i / |\mathcal{T}_i|$ is added to each of its outgoing neighbors' residuals ($0 < \alpha < 1$). Thus, any step decreases the residual by at least $\gamma(1-\alpha)$, and moves $\mathbf{x}$ closer to the solution. □

Theorem 2. *Algorithm 2 guarantees $\|\mathbf{r}\|_\infty < \epsilon$ when it is converged.*

Proof. Whenever a node's ATR is updated, the residual of each of its outgoing neighbors is increased. Thus, if we ever change a node's ATR, we need to add its outgoing neighbors to the `worklist` to verify that their residual is sufficiently small. This is what Algorithm 2 does. □

3.2 Residual-based Asynchronous ATR

Based on the analysis in Theorem 1, we note that the new ATR score of a node can be updated by just adding its current ATR score and its current residual by (2). To update the ATR scores in this way, we need to explicitly maintain the residual value for each node. Note that the residual of a node can be updated by (3). We design the residual-based asynchronous ATR algorithm shown in Algorithm 3 which is similar to the push-based data-driven PageRank in [20].

Let us compare Algorithm 3 and Algorithm 2. Whenever a node's ATR score is updated, all of its outgoing neighbors (on G') are pushed into the `worklist` in Algorithm 2. On the other hand, since we explicitly maintain the residual of each node in Algorithm 3, we can decide whether a node should be pushed to the `worklist` or not based on its residual. This can significantly reduce the unnecessary repeated computations.

Table 1: Real-world Web graph

No. of normal documents	437,386 (74.88%)
No. of spam documents	45,641 (7.81%)
No. of unlabeled documents	101,065 (17.30%)
No. of total documents	584,092
No. of links	2,470,557

4 EXPERIMENTAL RESULTS

We get a real-world Web graph from NAVER corporation which operates the most popular search engine called NAVER in Korea. We extract a subgraph of the entire Web graph using a variation of the forest fire graph sampling method [12]. Table 1 shows the basic statistics of the dataset. Among the 584,092 Web documents, 483,027 documents (82.7%) are labeled by human experts, i.e., those documents are manually classified into 'spam' or 'normal'.

We first test the performance of the Anti-TrustRank algorithm in terms of detecting spam documents. Let sync denote Algorithm 1, async denote Algorithm 2, and rasync denote Algorithm 3. As described in Section 2.1, human experts are supposed to manually label a subset of the nodes denoted by $\mathcal{L}$. Usually, the size of $\mathcal{L}$ is assumed to be very small since labeling requires human efforts. Among the nodes in $\mathcal{L}$, the set of spam documents $\mathcal{S}$ is considered to be the seed nodes in the Anti-TrustRank algorithm. In our experiments, we assume that p portion of the nodes can have labels among the entire vertex set $\mathcal{V}$. That is, the number of labeled documents $|\mathcal{L}| = p|\mathcal{V}|$.

When we finish the Anti-TrustRank computation, we order the documents in descending order according to the Anti-TrustRank scores. A document with a high ATR score indicates that the document is likely to be a spam document. When we pick top m documents, those documents are considered to be spam documents. Since the Anti-TrustRank algorithm computes a biased PageRank with the set $\mathcal{S}$, the Anti-TrustRank scores are propagated from the seed set $\mathcal{S}$. Thus, the number of documents associated with non-zero Anti-TrustRank scores is proportional to the size of $\mathcal{S}$.

We pick top $q|\mathcal{S}|$ documents (i.e., $m = q|\mathcal{S}|$) and count the number of spam documents, normal documents, and unlabeled documents among the retrieved documents. Table 2 shows the results with different p and q values. We set the tolerance parameter $\epsilon = 10^{-8}$, and notice that all the three methods, sync, async, and rasync, return the identical results for classifying the retrieved documents. We see that most of the retrieved documents are correctly classified into spam. This shows that in our dataset, the spam documents tend to be referred by other spam documents, which enables the Anti-TrustRank algorithm to work reasonably well.

Now, we investigate the computational cost of the sync, async, and rasync methods. By varying ϵ and p, we count the number of ATR updates and the number of arithmetic operations required to make each method converge. Table 3 shows the results.

Table 2: Accuracy of the retrieved documents

		$q = 1$	$q = 2$	$q = 3$	$q = 4$	$q = 5$
$p = 0.01$	No. of spam documents	1,367 (100%)	2,732 (99.927%)	4,099 (99.951%)	5,466 (99.963%)	6,833 (99.971%)
	No. of normal documents	0 (0%)	0 (0%)	0 (0%)	0 (0%)	0 (0%)
	No. of unlabeled documents	0 (0%)	2 (0.073%)	2 (0.049%)	2 (0.037%)	2 (0.029%)
$p = 0.02$	No. of spam documents	3,083 (100%)	6,030 (97.794%)	9,113 (98.530%)	12,196 (98.897%)	15,279 (99.117%)
	No. of normal documents	0 (0%)	107 (1.735%)	107 (1.157%)	107 (0.868%)	107 (0.694%)
	No. of unlabeled documents	0 (0%)	29 (0.470%)	29 (0.314%)	29 (0.235%)	29 (0.188%)
$p = 0.03$	No. of spam documents	3910 (100%)	7,683 (98.248%)	11,593 (98.832%)	15,503 (99.124%)	19,413 (99.299%)
	No. of normal documents	0 (0%)	107 (1.368%)	107 (0.912%)	107 (0.684%)	107 (0.547%)
	No. of unlabeled documents	0 (0%)	30 (0.384%)	30 (0.256%)	30 (0.192%)	30 (0.154%)

Table 3: No. of ATR updates and arithmetic operations

		$p = 0.01$			$p = 0.02$			$p = 0.03$		
ϵ		SYNC	ASYNC	RASYNC	SYNC	ASYNC	RASYNC	SYNC	ASYNC	RASYNC
10^{-8}	No. of ATR updates	2,336,368	20,361	20,361	2,336,368	20,625	20,625	2,336,368	20,628	20,628
	No. of arithmetics	24,442,660	8,424,970	2,516,097	24,449,524	10,715,238	2,703,627	24,452,832	10,716,065	2,703,630
10^{-10}	No. of ATR updates	2,336,368	20,427	20,427	2,336,368	20,999	20,999	2,336,368	21,002	21,002
	No. of arithmetics	24,442,660	14,164,081	2,982,452	24,449,524	17,591,721	3,262,017	24,452,832	17,592,548	3,262,020
10^{-12}	No. of ATR updates	2,920,460	39,483	39,483	2,920,460	39,801	39,801	2,920,460	39,804	39,804
	No. of arithmetics	30,553,325	17,845,604	3,284,207	30,561,905	25,816,773	3,932,402	30,566,040	25,817,600	3,932,405

We first notice that the asynchronous algorithms, ASYNC and RASYNC, make much fewer ATR updates than the synchronous algorithm, SYNC. This is because the asynchronous algorithms maintain a working set to selectively process the nodes whose ATR scores need to be updated while the synchronous algorithm processes all the nodes at every iteration. The number of ATR updates made in ASYNC and RASYNC should be the same because these methods follow the same rule to update the ATR scores.

In terms of the number of arithmetic operations, the asynchronous algorithms also save much computation compared to the synchronous algorithm. When we compare ASYNC and RASYNC, we see that the residual-based asynchronous algorithm (RASYNC) significantly reduces the number of arithmetic computations. As described in Section 3.2, RASYNC is able to effectively reduce the size of the working set by exploiting the problem structure, which results in filtering out unnecessary computations.

5 CONCLUSIONS & FUTURE WORK

We develop asynchronous Anti-TrustRank algorithms which are shown to be effective in reducing the number of computations compared to the synchronous Anti-TrustRank algorithm on a real-world Web graph. While achieving the same precision with the synchronous Anti-TrustRank method for Web spam detection, the asynchronous Anti-TrustRank algorithms require much fewer Anti-TrustRank updates as well as arithmetic operations than the synchronous method. We plan to investigate the distinguishing characteristics of Web spams to incorporate them into the spam ranking system. Also, we intend to apply the idea of the asynchronous personalized PageRank computation to other graph mining applications such as community detection [19] and clustering [18].

Acknowledgements. This research was supported by NAVER Corp. and by Basic Science Research Program through the NRF of Korea funded by MOE (2016R1D1A1B03934766 and NRF-2010-0020210) to JW. This work was also supported by US NSF grant IIS-1546452 to ID.

REFERENCES

[1] L. Becchetti, C. Castillo, D. Donato, R. Baeza-YATES, and S. Leonardi. 2008. Link Analysis for Web Spam Detection. *TWEB* 2, 1 (2008).
[2] P. Berkhin. 2005. A survey on PageRank computing. *Internet Mathematics* (2005).
[3] S. Brin and L. Page. 1998. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks and ISDN Systems* 30, 1-7 (1998).
[4] B. Davison. 2000. Recognizing Nepotistic Links on the Web. In *AAAI Workshop on Artificial Intelligence for Web Search*. 23–28.
[5] D. F. Gleich, L. Zhukov, and P. Berkhin. 2004. *Fast Parallel PageRank: A Linear System Approach.* Technical Report YRL-2004-038.
[6] Z. Gyongyi, P. Berkhin, H. Garcia-Molina, and J. Pedersen. 2006. Link Spam Detection Based on Mass Estimation. In *VLDB*. 439–450.
[7] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. 2004. Combating Web Spam with Trustrank. In *VLDB*. 576–587.
[8] G. Jeh and J. Widom. 2003. Scaling Personalized Web Search. In *WWW*.
[9] J. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632.
[10] V. Krishnan and R. Raj. 2006. Web Spam Detection with Anti-Trust Rank. In *ACM SIGIR Workshop AIRWEB*. 37–40.
[11] R. Lempel and S. Moran. 2001. SALSA: The Stochastic Approach for Link-structure Analysis. *ACM Transactions on Information Systems* 19, 2 (2001).
[12] J. Leskovec and C. Faloutsos. 2006. Sampling from Large Graphs. In *KDD*.
[13] Q. Liu, Z. Li, J. Lui, and J. Cheng. 2016. PowerWalk: Scalable Personalized PageRank via Random Walks with Vertex-Centric Decomposition. In *CIKM*.
[14] P. Lofgren, S. Banerjee, and A. Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *WSDM*. 163–172.
[15] F. McSherry. 2005. A Uniform Approach to Accelerated PageRank Computation. In *WWW*. 575–582.
[16] D. Nguyen, A. Lenharth, and K. Pingali. 2013. A Lightweight Infrastructure for Graph Analytics. In *SOSP*. 456–471.
[17] N. Spirin and J. Han. 2012. Survey on Web Spam Detection: Principles and Algorithms. *ACM SIGKDD Explorations Newsletter* 13, 2 (2012), 50–64.
[18] J. Whang, D. Gleich, and I. Dhillon. 2015. Non-exhaustive, Overlapping k-means. In *SDM*. 936–944.
[19] J. Whang, D. Gleich, and I. Dhillon. 2016. Overlapping Community Detection Using Neighborhood-Inflated Seed Expansion. *TKDE* 28, 5 (2016), 1272–1284.
[20] J. Whang, A. Lenharth, I. Dhillon, and K. Pingali. 2015. Scalable Data-driven PageRank: Algorithms, System Issues, and Lessons Learned. In *Euro-Par*.
[21] B. Wu and B. Davison. 2005. Identifying Link Farm Spam Pages. In *WWW*.
[22] B. Wu, V. Goel, and B. Davison. 2006. Propagating Trust and Distrust to Demote Web Spam. In *WWW Workshop MTW*.
[23] W. Xie, D. Bindel, A. Demers, and J. Gehrke. 2015. Edge-Weighted Personalized PageRank: Breaking A Decade-Old Performance Barrier. In *KDD*. 1325–1334.