Fast Asynchronous Anti-TrustRank for Web Spam Detection

Joyce Jiyoung Whang*[1], Yeonsung Jung[1], Inderjit S. Dhillon[2], Seonggoo Kang[3], and Jungmin Lee[3]
[1] Sungkyunkwan University (SKKU), [2] The University of Texas at Austin, [3] Naver Corporation.
ACM International Conference on Web Search and Data Mining (WSDM)
Workshop on MIS2: Misinformation and Misbehavior Mining on the Web, 2018

Main Contributions

- Asynchronous Anti-TrustRank algorithms
  - Significantly reduce the number of arithmetic operations compared to the traditional synchronous Anti-TrustRank algorithm
  - Without degrading the performance in detecting Web spams
- Convergence of the asynchronous Anti-TrustRank algorithms
- Experiments on a real-world Web graph indexed by NAVER which is the most popular search engine in Korea.

Notation

- $G' = (\mathcal{V}, \mathcal{E}')$: 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'.
- $\boldsymbol{P} \equiv \boldsymbol{D}^{-1}\boldsymbol{A}$ ($\boldsymbol{D}$ is the degree diagonal matrix)
- $\mathcal{Q}_i$: the set of incoming neighbors of node i on G'
- $\mathcal{T}_i$: the set of outgoing neighbors of node i on G'
- $\mathbf{x}$: a vector of the ATR scores, $\mathbf{r}$: a vector of the residuals
- $\mathbf{e}_S$: a vector with ones for the positions of the seed spam documents and zeros for other positions

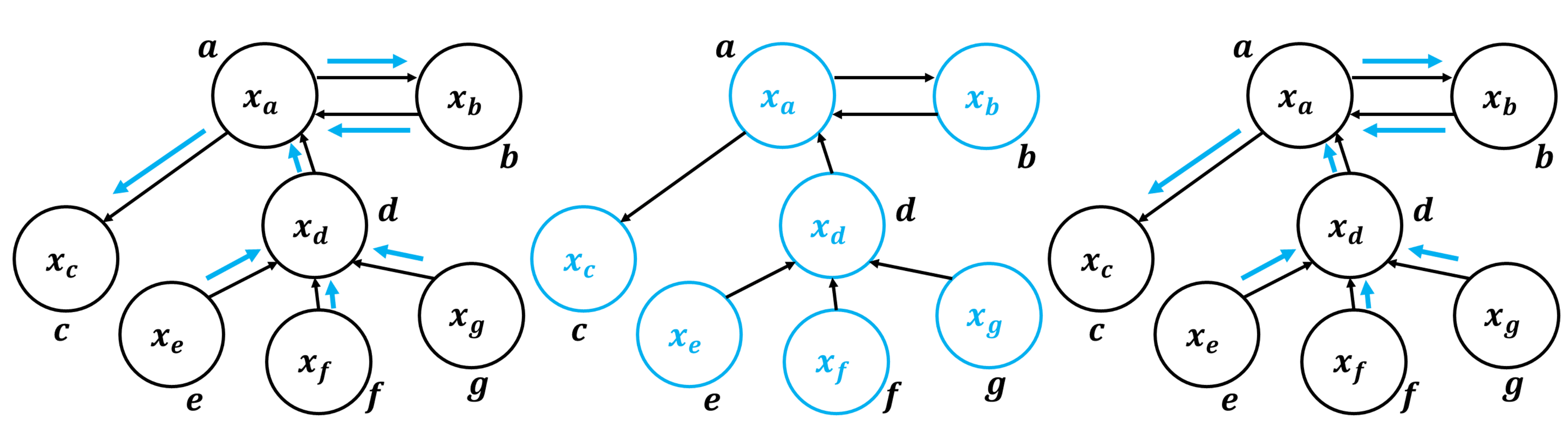
Anti-TrustRank*

- Spam pages are likely to be referred by other spam pages.
- Documents with high Anti-TrustRank (ATR) score → spam pages
- From spam 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.

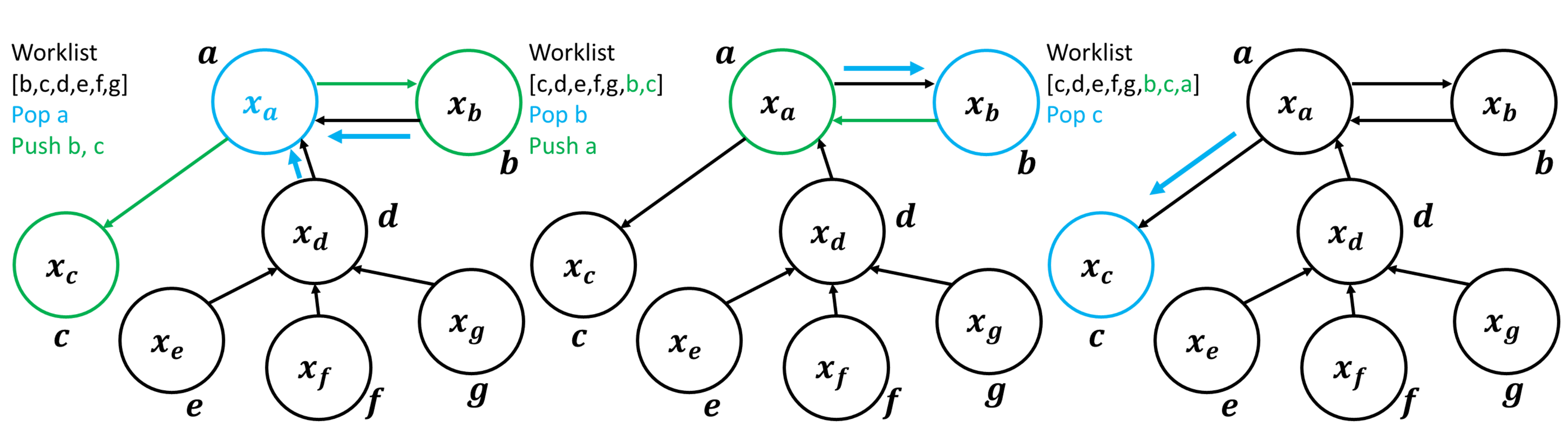
* V. Krishnan et al., Web spam detection with anti-trust rank. *AIRWeb*, 2006.

Algorithms

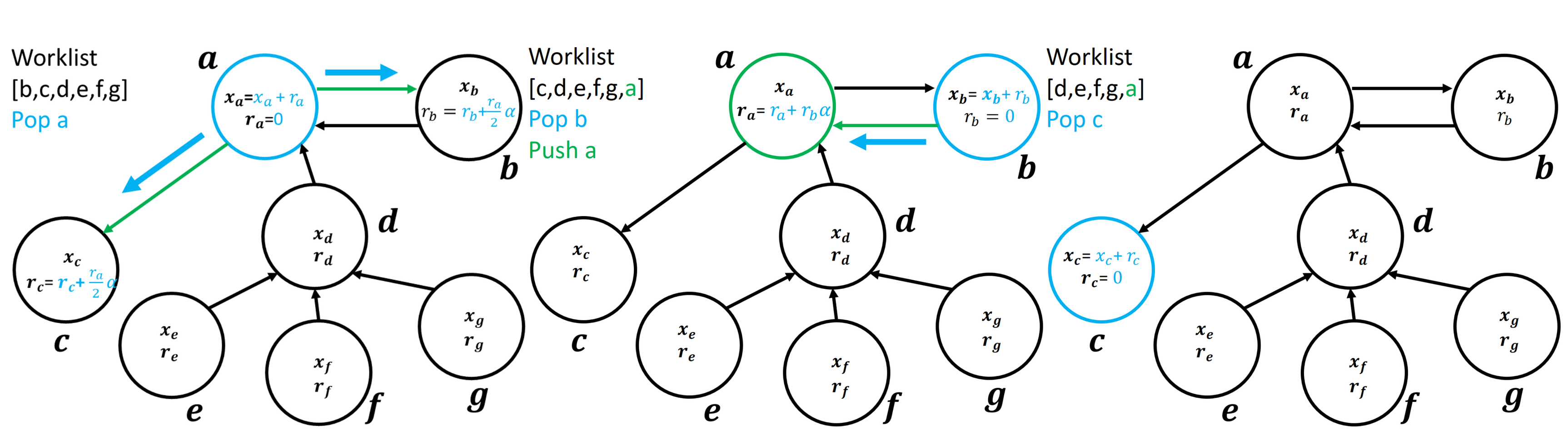
- Synchronous Anti-TrustRank (SYNC ATR)
  - The scores are updated after all the nodes re-compute the scores.



- Asynchronous Anti-TrustRank (ASYNC ATR)
  - worklist: a set of nodes whose ATR scores need to be updated.



- Residual-based Asynchronous Anti-TrustRank (RASYNC ATR)
  - new ATR = current ATR + current residual (explicitly maintain the residual of each node)
  - Filtering out unnecessary work in the `worklist`.



Pseudocodes

Algorithm: SYNC 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**
      $x_i^{new} = \alpha \sum_{j \in \mathcal{Q}_i} \frac{x_j}{|\mathcal{T}_j|} + (1-\alpha)$
    else
      $x_i^{new} = \alpha \sum_{j \in \mathcal{Q}_i} \frac{x_j}{|\mathcal{T}_j|}$
    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: ASYNC ATR

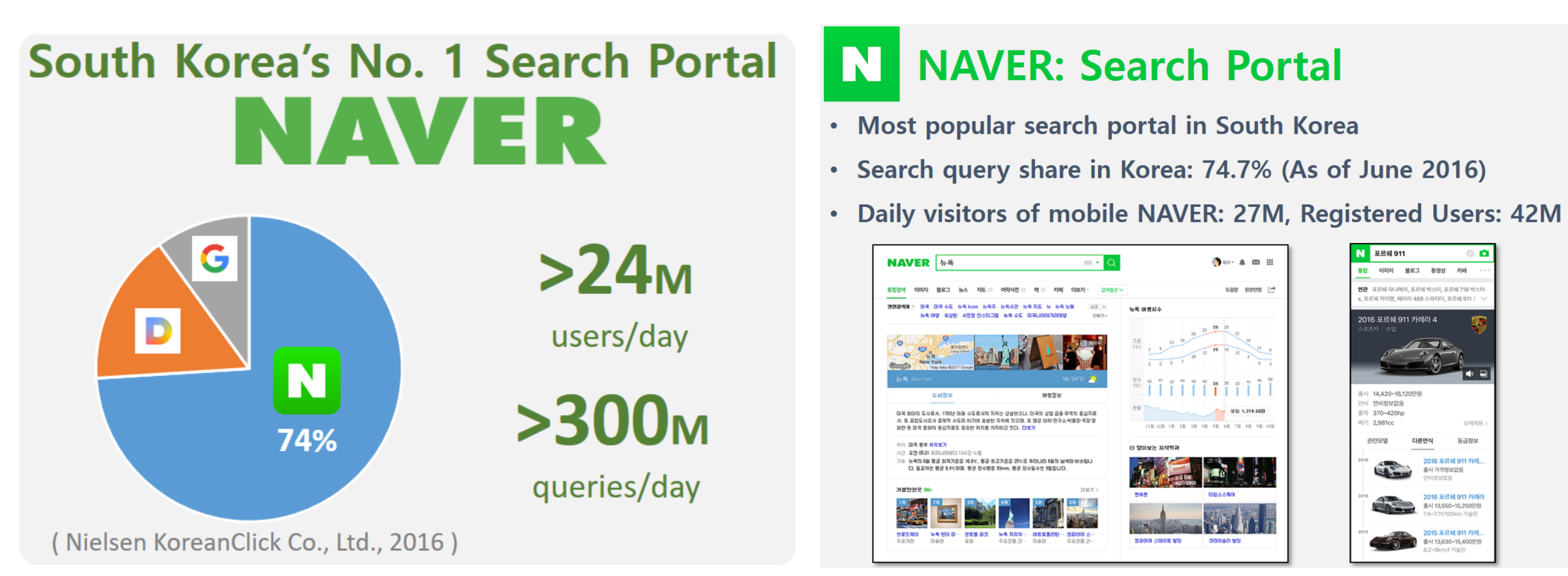
Input: $G' = (\mathcal{V}, \mathcal{E}')$, $\mathcal{S}$, α, ϵ
Output: ATR vector $\mathbf{x}$
Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}_S$
for $i \in \mathcal{V}$ **do**
  `wlist.push(i)`
end for
while `!wlist.empty` **do**
  i = `wlist.pop()`
  if $i \in \mathcal{S}$ **then**
    $x_i^{new} = \alpha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

Algorithm: RASYNC 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**
  `wlist.push(i)`
end for
while `!wlist.empty` **do**
  i = `wlist.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**
      `wlist.push(j)`
    end if
  end for
  $r_i = 0$
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

Experimental Results

- Real-world Web graph from NAVER corporation
  - 584,092 documents and 2,470,557 edges
  - 437,386 (74.88%) normal docs and 45,641 (7.81%) spam docs
  - 101,065 (17.30%) documents are unlabeled.



- Most of the retrieved documents are correctly classified into spam.
  - $|\mathcal{L}| = p|\mathcal{V}|$ where $\mathcal{L}$ denotes the set of labeled documents
  - Pick top $q|\mathcal{S}|$ documents where $\mathcal{S}$ denotes the set of spam seeds

Table: Accuracy of the retrieved documents

		$q=1$	$q=3$	$q=5$
$p=0.01$	spam docs	1,367 (100%)	4,099 (99.951%)	6,833 (99.971%)
	normal docs	0 (0%)	0 (0%)	0 (0%)
	unlabeled docs	0 (0%)	2 (0.049%)	2 (0.029%)
$p=0.02$	spam docs	3,083 (100%)	9,113 (98.530%)	15,279 (99.117%)
	normal docs	0 (0%)	107 (1.157%)	107 (0.694%)
	unlabeled docs	0 (0%)	29 (0.314%)	29 (0.188%)
$p=0.03$	spam docs	3910 (100%)	11,593 (98.832%)	19,413 (99.299%)
	normal docs	0 (0%)	107 (0.912%)	107 (0.547%)
	unlabeled docs	0 (0%)	30 (0.256%)	30 (0.154%)

- The asynchronous algorithms, ASYNC and RASYNC, make much fewer ATR updates than the synchronous algorithm, SYNC.
- RASYNC significantly reduces the number of arithmetic computations.

Table: No. of ATR updates and arithmetic operations

p	ϵ		SYNC	ASYNC	RASYNC
0.01	10^{-8}	No. of ATR updates	2,336,368	20,361	20,361
		No. of arithmetics	24,442,660	8,424,970	2,516,097
	10^{-12}	No. of ATR updates	2,920,460	39,483	39,483
		No. of arithmetics	30,553,325	17,845,604	3,284,207
0.03	10^{-8}	No. of ATR updates	2,336,368	20,628	20,628
		No. of arithmetics	24,452,832	10,716,065	2,703,630
	10^{-12}	No. of ATR updates	2,920,460	39,804	39,804
		No. of arithmetics	30,566,040	25,817,600	3,932,405