Scalable Anti-TrustRank with Qualified Site-level Seeds for Link-based Web Spam Detection

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

Yeonsung Jung
Sungkyunkwan University (SKKU)
Suwon, South Korea
ys.jung@skku.edu

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

Dongho Yoo
Naver Corporation
Seongnam, South Korea
dongho.yoo@navercorp.com

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

ABSTRACT

Web spam detection is one of the most important and challenging tasks 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. In this paper, we conduct a comprehensive analysis of the link structure of web spam using real-world web graphs to systemically investigate the characteristics of the link-based web spam. By exploring the structure of the page-level graph as well as the site-level graph, we propose a scalable site-level seeding methodology for the Anti-TrustRank (ATR) algorithm. The key idea is to map a website into a feature space where we learn a classifier to prioritize the websites so that we can effectively select a set of good seeds for the ATR algorithm. This seeding method enables the ATR algorithm to detect the largest number of spam pages among the competitive baseline methods. Furthermore, we design work-efficient asynchronous ATR algorithms which are able to significantly reduce the computational cost of the traditional ATR algorithm without degrading the performance in detecting spam pages while guaranteeing the convergence.

CCS CONCEPTS

• **Information systems** → **Spam detection**; *Web searching and information discovery*; *Web search engines.*

KEYWORDS

Web Spam Detection, Anti-TrustRank, Seeds, Link Analysis.

ACM Reference Format:
Joyce Jiyoung Whang, Yeonsung Jung, Seonggoo Kang, Dongho Yoo, and Inderjit S. Dhillon. 2020. Scalable Anti-TrustRank with Qualified Site-level Seeds for Link-based Web Spam Detection. In *Companion Proceedings of the Web Conference 2020 (WWW '20 Companion), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3366424.3385773

*Corresponding author

WWW '20 Companion, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7024-0/20/04.
https://doi.org/10.1145/3366424.3385773

1 INTRODUCTION

Web spam detection is an important task in web search. Given a web graph with a set of nodes and edges where a node indicates a web page and an edge indicates a hyperlink between the pages, search engines exploit the link structure to rank and prioritize the users' search results [7], [17]. Web spam refers to web pages that confuse and mislead the search engines to get higher-than-deserved rankings. A number of web spam detection methods have been proposed including link-based methods [27] [10], content-based methods [5], and hybrid methods [25]. In particular, link analysis has been considered to be an important feature of a good spam detection method [3]. Even though a number of link-based spam detection methods have been proposed, most existing link-based methods focus on some specific properties of link spam instead of generalizing and utilizing the diverse characteristics of link spam. For example, [33] has been proposed to detect a link farm which is one of the well-known structures of link spam whereas [9] exploits the clusterable structure of web spam.

We propose a practical link-based web spam detection method which is based on a systematic analysis of the structure of real-world web graphs crawled by the NAVER search engine (www.naver.com) which is the most popular search engine in South Korea. We model the web using two different types of graphs – the page-level graph where a node indicates a web page and the site-level graph where a node indicates a website. Indeed, a thorough investigation of the real-world web graphs allows us to confirm that the Anti-TrustRank (ATR) algorithm [18] is a useful method to detect real-world link spam at the page-level graph because spam pages are likely to be referred by other spam pages. We discuss the advantages of the ATR method over other link-based spam detection methods and how our approaches can be incorporated into other fancier spam detection methods (e.g., [5], [28]) in practice in Section 8.

Even though how to select seeds in the ATR method plays the most important role in the success of the ATR algorithm, most existing seeding methods fail to scale to large-scale web graphs due to the sparsity of the solution, e.g., PageRank-based [16] seeds and topic-based seeds [34]. We propose a scalable site-level seeding methodology where we represent a website as a feature vector and learn a classifier to predict the probability of being spam. By prioritizing the websites based on the probability, we identify a set of good seeds which are expected to effectively propagate their

ATR scores to the spam pages. We observe that our seeding method enables the ATR algorithm to detect the largest number of spam pages and achieve the highest F1 score among the competitive baseline methods.

The success of our method is based on a two-level analysis of link structure, i.e., the site-level and the page-level analysis. While most existing spam detection methods focus either on classifying sites (e.g., [9], [26]) or individual web pages (e.g., [16]), we integrate these two ideas into one framework to develop a scalable spam detection method. Indeed, our methodology can be a practical solution for a real-world web spam detection problem where we have a limited budget on labeling the nodes in a large-scale web graph. In our experiments, we reflect this realistic constraint, and it is encouraging that our method outperforms other state-of-the-art methods in this setting. Details are provided in Section 7.1.

Furthermore, we reduce the computational cost of the ATR algorithm by proposing asynchronous ATR algorithms which significantly reduce the number of arithmetic operations while guaranteeing the convergence. Our residual-based asynchronous ATR algorithm is much faster than the traditional synchronous ATR algorithm without degrading performance in detecting spam pages.

Our main contributions can be summarized as follows.

- We explore the real-world web graphs (Section 2). While most existing spam detection methods consider a single web graph, we model the web using the page-level graph as well as the site-level graph. This two-level analysis enables us to generalize and explain the structure of link spam in the real-world datasets (Section 3).
- We propose an effective and scalable site-level seeding methodology for the ATR algorithm (Section 4).
- We propose efficient asynchronous ATR algorithms and prove the convergence (Section 5).
- We show that our seeding method allows the ATR algorithm to detect the largest number of spam pages and our asynchronous ATR methods significantly reduce the computational cost of the traditional ATR algorithm (Section 7).

2 REAL-WORLD WEB GRAPHS AND THE SITE-LEVEL EXAMINATION

We get two real-world web graphs[1] by sampling the original web graph crawled by the NAVER search engine. We use a variation of the forest fire graph sampling method [20] to get the two real-world datasets shown in Table 1. For each of the datasets W1 and W2, we have two different types of graphs – the page-level graph (denoted by G) and the site-level graph (denoted by H). Since a website consists of a set of web pages, we can construct a site-level graph by representing each site as a node and adding a directed edge between two sites if a web page in one site has a hyperlink to a web page in the other site. We remove self-loops in G and H. All the nodes in our datasets W1 and W2 were labeled by human experts (i.e., the nodes are labeled as spam or normal). We notice that some sites were closed after the web pages in the sites had been crawled, and thus we cannot assign the labels to those. These nodes are denoted by undefined nodes. In Table 1, labeled edges indicate the edges whose both endpoints have labels.

[1] All our datasets and codes are available at bigdata.cs.skku.edu/down/ATR_2020.zip. We notice that some benchmark datasets are available on [8] and [4], but these datasets only include a host-level graph, and do not include a page-level graph. We were not able to test our methods on these benchmark datasets since our method incorporates both of the page-level and the site-level graphs, which leads to a more realistic and scalable solution for a large-scale web spam detection problem.

Table 1: Two Real-World Web Graphs

		page-level graph G	site-level graph H
W1	No. of normal nodes	797,718 (93.15%)	39,809 (68.63%)
	No. of spam nodes	47,301 (5.52%)	7,954 (13.71%)
	No. of undefined nodes	11,385 (1.33%)	10,239 (17.66%)
	No. of total nodes	856,404	58,002
	No. of labeled edges	3,929,401 (99.33%)	83,351 (85.67%)
	No. of edges	3,955,939	97,294
W2	No. of normal nodes	797,018 (91.20%)	39,984 (67.32%)
	No. of spam nodes	65,259 (7.47%)	8,846 (14.89%)
	No. of undefined nodes	11,684 (1.34%)	10,561 (17.78%)
	No. of total nodes	873,961	59,391
	No. of labeled edges	3,952,584 (99.33%)	84,373 (85.68%)
	No. of total edges	3,979,280	98,478

It took around three months for ten human experts to label the nodes in W1 and W2. To reduce noise in the labeling process, senior engineers cross-checked the node labels. They were able to label the large graphs quickly using some tricks explained below, and indeed, those tricks are the important heuristics adopted in a large search engine company where a set of human-labeled seeds is considered to be a part of important input of a web spam detection system while there is a limited budget on the human labeling process.

When human experts label the pages and the sites, they can use some tricks to speed up the labeling process. The key here is to perform a site-level examination followed by the refinement of the page labels. This site-level examination is much more efficient and scalable than a page-level examination (i.e., looking at all individual pages) in that it requires less number of examinations to label the same number of pages. Let us provide the details about the site-level examination. Basically, the human experts are supposed to examine individual sites instead of individual pages. If a website is a spam site, the human experts can label the site as spam by just looking at a few pages inside the site. If a site turns out to be spam, all the pages in the spam site are considered to be spam pages (by definition of a spam site). On the other hand, it is not guaranteed that a normal site always contains only normal pages because a subset of pages inside a normal site might have been polluted by spammers while the rest of the pages are normal. Accordingly, it may take a little more effort to confirm that a page is normal. Even after the human experts decide to label a site as normal, they need to examine more pages to check whether the site contains any spam pages. If there exists a subset of spam pages inside a normal site, the human experts label those pages as spam. This refinement step can be done in an efficient way because the human experts exploit the structure of the URL to label the spam pages.

3 A TWO-LEVEL ANALYSIS OF LINK SPAM

Using the real-world web graphs described in Section 2, we investigate the link structure of web spam on the page-level graph as well as the site-level graph. Even though there has been extensive research on link spam [27], most existing approaches focus on either a page-level graph or a site-level graph, and do not consider both

Table 2: Different Types of Edges on W1

		$\lvert\mathcal{E}\rvert$	$E(\lvert\mathcal{E}\rvert)$	conclusion	p-value
G	normal → normal	3,639,884	3,500,494	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	7.0×10^{-23}
	normal → spam	2,157	208,725	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	7.9×10^{-28}
	spam → normal	73,049	207,807	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	7.2×10^{-55}
	spam → spam	214,311	12,375	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	9.2×10^{-63}
H	normal → normal	56,647	57,840	$\lvert\mathcal{E}\rvert \neq E(\lvert\mathcal{E}\rvert)$	2.6×10^{-2}
	normal → spam	17,551	11,771	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	5.6×10^{-13}
	spam → normal	4,394	11,418	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	9.1×10^{-28}
	spam → spam	4,759	2,321	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	9.2×10^{-21}

Table 3: Types of the Between-Site Edges on W1

Source		Destination		$\lvert\mathcal{E}\rvert$	$E(\lvert\mathcal{E}\rvert)$	conclusion	p-value
Site	Page	Site	Page				
Normal	Normal	Normal	Normal	857,565	666,284	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	2.0×10^{-20}
Normal	Normal	Normal	Spam	13	39,750	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	5.5×10^{-17}
Normal	Normal	Spam	Spam	1,205	5,611	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	5.1×10^{-10}
Normal	Spam	Normal	Normal	10,825	39,562	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	9.8×10^{-32}
Normal	**Spam**	**Normal**	**Spam**	52,392	2,357	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	4.9×10^{-55}
Normal	**Spam**	**Spam**	**Spam**	121,397	336	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	1.7×10^{-85}
Spam	Spam	Normal	Normal	5,953	7,361	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	1.3×10^{-5}
Spam	Spam	Normal	Spam	340	453	$\lvert\mathcal{E}\rvert < E(\lvert\mathcal{E}\rvert)$	2.6×10^{-3}
Spam	**Spam**	**Spam**	**Spam**	3,768	67	$\lvert\mathcal{E}\rvert > E(\lvert\mathcal{E}\rvert)$	2.0×10^{-52}

of the graphs. By analyzing the characteristics of link spam on the two different levels of graphs, we generalize the structure of link spam, which allows us to design practical solutions for large-scale web spam detection problems which will be discussed in Section 4.

3.1 Edge Classification at the Page-level Graph and the Site-level Graph

As described in Table 1, we have node labels (either normal or spam) on both G and H. We classify the edges into four categories based on the labels of their endpoints as shown in Table 2. To check whether there exist some strong interactions between particular types of the nodes, we count the number of edges for each category and compare it with the number of edges in a randomly shuffled graph where we randomly shuffle the node labels while preserving the number of each type of the node labels. In Table 2, $\lvert\mathcal{E}\rvert$ indicates the actual number of edges in G and H, and $E(\lvert\mathcal{E}\rvert)$ indicates the expected number of edges when the edges are randomly organized. For $E(\lvert\mathcal{E}\rvert)$, we report the average number of edges over 30 different random shuffles. To check whether the difference between $\lvert\mathcal{E}\rvert$ and $E(\lvert\mathcal{E}\rvert)$ is statistically significant, we conduct a t-test, and report the p-value and the conclusion with the confidence level of 98%. We only show the results on W1 because we have similar results on W2. At the page-level graph G, we see that normal pages tend to point to other normal pages, which is the idea of the TrustRank algorithm [16]. Also, spam pages tend to be referred by other spam pages, which is the principle of the Anti-TrustRank method [18]. These results show that the TrustRank and the Anti-TrustRank algorithms might work reasonably well at the page-level graph.

On the other hand, at the site-level graph H, it is interesting to see that the number of edges from normal nodes to spam nodes is also significant as well as the edges from spam nodes to spam nodes. This indicates that at the site-level graph, the spam sites can be pointed by either normal sites or spam sites. To analyze what kind of configurations can make these types of edges at H, we take a deeper look at the between-site edges. When we consider an incident node of a between-site edge, we can think of three cases: (i) the site is normal and the page is normal, (ii) the site is normal but the page is spam, (iii) the site is spam and the page is spam. By definition of a spam site, the pages within a spam site are considered to be spam pages. Recall the labeling process in Section 2. Thus, given a between-site edge, there can be nine different types of edges based on the labels of the source node and the destination node as shown in Table 3. We observe three significant edge types: NSNS, NSSS, SSSS (we focus on the edges which are incident to spam nodes). In particular, the NSSS (Normal site, Spam page, Spam site, Spam page) type is the most significant pattern. From the site-level view, this edge type explains the connection from normal nodes to spam nodes, and thus, we note that the strong connections from normal to spam observed in Table 2 are mainly due to the interactions between spam pages in normal sites and those in spam sites. Note that, from the page-level view, all the three significant types of edges (i.e., NSNS, NSSS, SSSS) are the edges from spam pages to spam pages. In what follows, we present the dominant patterns of link spam in the real-world datasets, which explain the three most significant edge types observed in Table 3.

3.2 Types of Link-based Web Spam: Two-level Edge Classification Perspective

The two-level analysis of link structure in Section 3.1 allows us to figure out dominant patterns of link spam from the page-level and the site-level viewpoints. By exploring our real-world web graphs, we observe three dominant configuration types of web spam as shown in Figure 1 where a blue rectangle indicates a normal site and a red rectangle indicates a spam site whereas a blue circle indicates a normal page and a red circle indicates a spam page.

Overpost. As shown in Figure 1(a), a spammer makes a lot of postings in different normal sites to intrigue transactions into the targeting spam site. Obviously, the postings are spam pages which contain the links to the spam pages in the spam site. This configuration makes the NSSS edge type described in Table 3.

Hacking. Figure 1(b) shows the case where a spammer hacks normal sites. The spammer makes spam pages in normal sites and the spam pages are linked to other spam pages. There can be two different types of hacking: (i) the interactions between the spam pages involve a spam site – the upper black triangle in Figure 1(b), (ii) the interactions between the spam pages only involve normal sites – the lower black triangle in Figure 1(b). The latter indicates that the spam content exists in one of the normal sites, and thus there is no need to have a link to a spam site. In this 'Hacking' configuration type, we can observe the NSSS and NSNS edges shown in Table 3.

Link Farm. Some spam sites and spam pages are designed to be densely connected with each other to raise PageRank scores so that they can be indexed by a search engine. Figure 1(c) shows this structure. This type of configuration is called as a link farm [33] where we can observe the SSSS edge type in Table 3.

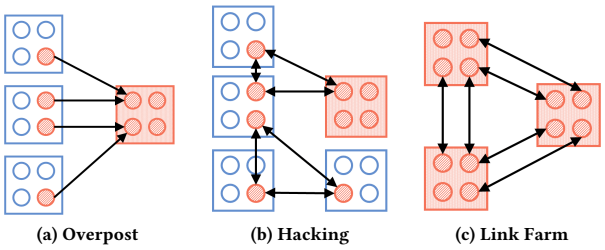


Figure 1: Three Dominant Types of Link Spam. A blue rectangle represents a normal site and a red rectangle represents a spam site. A blue circle represents a normal page and a red circle represents a spam page.

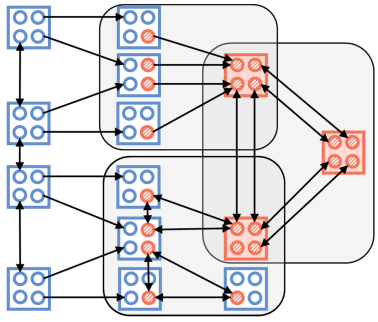


Figure 2: A Possible Combination of the Dominant Patterns of Link Spam

Some variations of the above configurations might have been studied in various contexts [27]. However, most existing approaches do not simultaneously consider the page-level and the site-level analysis, and tend to describe the structure of link spam in an ad hoc manner. Our contribution is to define explainable building blocks of link spam by the two-level link analysis. For example, Figure 2 shows a possible real-world link spam which can be explained by a combination of the aforementioned building blocks.

4 ANTI-TRUSTRANK WITH QUALIFIED SITE-LEVEL SEEDS

From a search engine perspective, there should be a metric to determine whether a *page* is spam or not. Based on our analysis in Section 3, we note that the Anti-TrustRank (ATR) algorithm [18] might successfully detect spam pages at the page-level graph because we observe that spam pages are likely to be referred by other spam pages. The Anti-TrustRank is a label propagation method where the labels are propagated from carefully selected seeds. How to select the seeds plays a critical role in the success of the ATR algorithm. It is important to note that the seeds should be examined by human experts to get the labels. To accelerate this labeling process, we conduct a site-level examination instead of a page-level examination in practice as described in Section 2. Thus, a key challenge here is that we should select good seeds for the page-level propagation while we are only able to conduct a site-level examination. In this section, we present our solutions for this problem based on the structural relationship between the page-level and the site-level graphs described in Section 3.

4.1 The Anti-TrustRank Algorithm

Let us briefly describe the ATR algorithm. Given a graph $G = (\mathcal{V}, \mathcal{E})$, an ATR score is assigned to every node in the graph such that a node with a high ATR score indicates that the node is likely to be a spam page. To run the ATR algorithm, we should select seeds from which the ATR scores start to spread to other nodes in the graph. Note that the seeds should be spam pages. However, we do not know where the spam pages are placed in advance. To determine the spam seeds, we first need to select a set of candidates, denoted by $\mathcal{L}$ which will be examined by human experts (where $\mathcal{L} \subset \mathcal{V}$). Note that $\mathcal{L}$ might contain normal nodes as well as spam nodes. After the examination, only spam nodes can be used as the spam seeds for the ATR algorithm among the nodes in $\mathcal{L}$. Since the labeling process requires human efforts, the size of $\mathcal{L}$ is usually much smaller than the size of $\mathcal{V}$, which indicates that how to choose $\mathcal{L}$ critically affects the performance of the ATR algorithm. For example, if we have a *bad* $\mathcal{L}$, we might end up with failing to find any spam nodes in $\mathcal{L}$, which means that we cannot run the ATR algorithm appropriately.

Once we determine the spam seeds, the ATR scores of the spam seeds are initialized to be ones whereas the ATR scores of the rest of the nodes are initialized to be zeros. The ATR scores are propagated to incoming neighbors of the spam seeds so that the nodes having links to spam nodes end up with having high ATR scores. Indeed, computing the ATR scores can be interpreted as computing a personalized PageRank (also called as a biased PageRank) on the reverse graph $G' = (\mathcal{V}, \mathcal{E}')$ where $\mathcal{E}'$ indicates the set of reverse edges (i.e., if an edge $\{i, j\} \in \mathcal{E}$ then $\{j, i\} \in \mathcal{E}'$) with the spam seeds as the teleportation (or personalization) set.

4.2 Qualified Site-level Seeds

To apply the ATR algorithm to large-scale web graphs, we should select $\mathcal{L}$ such that (1) it includes as many spam nodes as possible, and (2) it includes spam nodes whose ATR scores can effectively propagate to other spam nodes in the graph. Also, since human experts conduct a site-level examination as described in Section 2 (i.e., the pages in $\mathcal{L}$ are grouped by the sites), we should design a site-level seeding strategy which satisfies the above objectives.

4.2.1 Feature Representations for Sites. To construct a site-level seed set, we propose to model each site as a feature vector and build a classifier that predicts the probability of being spam so that we can assign higher priority to the sites that are likely to be spam when we construct $\mathcal{L}$. That is, by mapping the sites into a feature space, we train a classifier that predicts the label of the sites with small training data, e.g., 10% of the sites (since labeling sites requires human efforts, 10% might be a practical upper bound). Then, we sort the sites according to the probability of being spam to include the top-ranked sites in $\mathcal{L}$.

In this process, how to represent a site as a feature vector is an important issue. The features should be able to appropriately discriminate the spam sites from the normal sites. We define 16 features as shown in Table 4 where $\bar{H}$ denotes the undirected version of H and $\bar{H}_w$ denotes the undirected, weighted site-level graph. For

Table 4: Our Features to Model a Site

in-p: indegree of each page in the site h	
out-p: outdegree of each page in the site h	
dist: the distances from the site h to all other reachable sites on $\bar{H}$	
entro-in-p: entropy of ***in-p***	**reachability**: no. of reachable sites on $\bar{H}$
entro-out-p: entropy of ***out-p***	**cluster**: whether h belongs to a spam cluster
mean-dist: mean of ***dist***	**dmnt-ratio**: max. weight/degree of h on $\bar{H}_w$
std-dist: standard deviation of ***dist***	**no-page**: no. of pages in the site h
max-dist: maximum of ***dist***	**in-page**: no. of pages having an edge to h
within-site: no. of within-site edges	**out-page**: no. of pages having an edge from h
in-h: indegree of the site h on H	**one-hop**: no. of one-hop distant sites on $\bar{H}$
out-h: outdegree of the site h on H	**two-hop**: no. of two-hop distant sites on $\bar{H}$

example, **entro-in-p** considers the entropy of the indegree of each page in a site h. Since spam pages tend to be referred by machine-generated pages, each page in a spam site might have the similar number of inlinks from the outside of the site, which leads to a low entropy value. On the other hand, in a normal site, usually there are a few popular pages that receive a lot of inlinks while the rest of the pages have a few inlinks, which leads to a high entropy value. Thus, the **entro-in-p** might be a useful feature to differentiate the spam sites from the normal sites. The **cluster** feature in Table 4 indicates whether a site belongs to a spam cluster or not. To extract this feature, we perform graph clustering [32] on the site-level graph, and if more than half of the nodes in a cluster are spam nodes (we assume that we only know the labels of nodes in a training set), we consider the cluster to be a spam cluster. If a site belongs to a spam cluster, the value of the **cluster** feature is set to be one. If a site does not belong to a spam cluster, the feature value is set to be zero. This idea is similar to that of [9] where the host labels are refined based on graph clustering. While [9] exploits the clusterable structure of link spam [33] in a post-processing step, we directly encode it as one of the features to represent a site. Some of the features presented in Table 4 might include a subset of features studied in a host classification problem[2] [26]. Even though we cannot explain each of the 16 features in detail because of lack of space, we empirically show the usefulness of our features in Section 4.2.3.

4.2.2 Relabeling Hacked Sites. The goal of the site-level seeding is to efficiently select good spam seeds for the ATR algorithm so that the ATR scores are well spread to the spam nodes. Let us review the structure of link spam in Section 3.2. In Figure 1, if we use the pages in the spam sites as the spam seeds, we can detect all the spam pages (when we propagate the ATR scores to the incoming neighbors of the seeds) except the spam pages which are involved in the lower triangle of the hacking type shown in Figure 1(b). Since all the sites involved in this lower triangle are normal sites, our classifier-based seeding might not consider any of these sites as seeds, which results in failing to detect the hacked spam pages (note that as long as any of the spam pages in this triangle is not included in the seeds, the ATR algorithm fails to detect these spam pages). Therefore, we relabel the hacked sites as spam (instead of normal) in the training phase so that the hacked sites can be considered as spam sites, which allows us to include the hacked pages as the spam seeds for the ATR algorithm. We observe that this relabeling process leads to improving the performance of the ATR algorithm.

[2] webspam.lip6.fr/wiki/pmwiki.php?n=Main.PhaseI

Table 5: Classification Performance of the Features

	W1		W2	
	node2vec	Our Features	node2vec	Our Features
Accuracy	83.9%	**88.0%**	82.7%	**88.1%**
Normal F1	90.6%	**92.1%**	89.7%	**92.2%**
Spam F1	46.1%	**86.1%**	45.1%	**86.1%**
Avg. Precision	70.5%	**88.8%**	70.2%	**89.0%**
Avg. Recall	66.8%	**89.4%**	65.7%	**89.3%**
Avg. F1	68.3%	**89.1%**	67.4%	**89.1%**

4.2.3 Classification Performance. We test the performance of the site-level classification to show that our features described in Section 4.2.1 are useful. Among well-known classifiers implemented in Scikit-learn library [24], the random forest [6] shows the best performance. In Table 5, we compare our features with the node2vec features [14] in terms of the classification performance. The node2vec method is a state-of-the-art node embedding method that is able to compute a mapping of nodes to low-dimensional feature vectors. Even though we know that the parameters of the node2vec method might be further finely tuned, we note that our features show better performance than the node2vec features, and our features show reasonable performance in predicting spam sites.

4.2.4 Qualified Site-level Seeds. Once we learn a classifier to get the probability of being spam for each site, we construct $\mathcal{L}$ by taking top-ranked sites that are likely to be spam (note that the hacked sites are also included in this process due to the relabeling process described in Section 4.2.2). These sites are examined by human experts as described in Section 2 so that the sites and the pages are labeled. Finally, the spam pages in $\mathcal{L}$ are used as spam seeds in the ATR algorithm. Let $\mathcal{S}$ denote the set of spam seeds ($\mathcal{S} \subseteq \mathcal{L}$). We expect that, with our qualified site-level seeds, the size of $\mathcal{S}$ is large and the nodes in $\mathcal{S}$ are good seeds to detect the spam pages in the page-level graph.

5 WORK-EFFICIENT ANTI-TRUSTRANK

Once we get the spam seeds, the ATR scores should be computed on the page-level graph which usually consists of more than tens of billions of nodes. We propose efficient ATR algorithms which significantly increase the scalability of the traditional ATR algorithm.

5.1 Synchronous Anti-TrustRank

As described in Section 4.1, computing the ATR scores is identical to computing the personalized PageRank on the reverse graph with $\mathcal{S}$ as the personalization set. Let $\boldsymbol{A}$ denote the adjacency matrix of G' where G' denotes the graph with reverse edges. Let $\mathcal{Q}_i$ 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 spam seeds (i.e., the nodes in $\mathcal{S}$) and zeros for the 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. Algorithm 1 describes the traditional synchronous ATR algorithm where the ATR scores are updated only after all the nodes recompute the ATR scores.

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 $\|\boldsymbol{\delta}\|_\infty < \epsilon$ **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**
  Add i to workqueue
end for
while !workqueue.isEmpty **do**
  Take i from workqueue
  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 workqueue **then**
        Add j to workqueue
      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 $r_i \geq \epsilon$ **do**
  Add i to workqueue
end for
while !workqueue.isEmpty **do**
  Take i from workqueue
  $x_i^{new} = x_i + r_i$
  $x_i = x_i^{new}$
  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**
      Add j to workqueue
    end if
  end for
  $r_i = 0$
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

5.2 Asynchronous Anti-TrustRank

Instead of updating the ATR scores of all the nodes at every iteration, we can design an asynchronous ATR algorithm where we manage a workqueue that contains a set of nodes whose ATR scores need to be updated[3]. The idea is that some nodes in a graph might require frequent updates to get the converged ATR scores while other nodes might require only a few updates since the degree distribution of real-world web graphs follows a power-law. Therefore, if we focus more on the nodes that require more computations instead of equally updating the ATR scores of all the nodes, we might be able to compute the ATR scores more efficiently.

Algorithm 2 shows how we can implement this idea. Initially, the workqueue contains all the vertices. From the workqueue, we take a node at a time, then compute the ATR score of the node. If the difference between the newly computed ATR score and the current ATR score is greater than or equal to the tolerance ϵ, then we update its ATR score. Once a node's ATR score is updated, we should recompute the ATR scores of its outgoing neighbors on G' (i.e., incoming neighbors on G). Thus, whenever we process a node from the workqueue, we add the outgoing neighbors of the processed node (on G') to the workqueue. We repeat this process until the workqueue becomes empty. For the global PageRank problem [7], a similar approach has been explored in [31] where a data-driven PageRank has been studied. Algorithm 2 can be considered as an extension of this idea to the ATR computation where we have a specialized personalization set for the spam detection task. By extending the analysis of [23], we theoretically show the convergence of Algorithm 2 as shown in Theorem 1.

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

[3] A preliminary version of the asynchronous ATR algorithm has been presented in [30] without any official proceedings.

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 workqueue, we update its ATR value as follows:

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

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

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

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

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

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

Thus, we see that

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

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

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

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. □

Also, Theorem 2 shows that the converged solution of Algorithm 2 guarantees $\|\mathbf{r}\|_\infty < \epsilon$ where ϵ denotes the tolerance.

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

Proof. When a node's ATR is updated, the residual of each of its outgoing neighbors is increased. Thus, whenever we change a node's ATR, we need to add its outgoing neighbors to the `workqueue` to verify that their residuals are sufficiently small, i.e., their residuals are smaller than ϵ. This is what Algorithm 2 does. □

5.3 Residual-based Asynchronous Anti-TrustRank

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. Let us compare Algorithm 3 and Algorithm 2. In Algorithm 2, whenever we update a node's ATR score, we blindly add all of its outgoing neighbors (on G') into the `workqueue` to verify that their residuals are less than the tolerance. On the other hand, in Algorithm 3, we explicitly maintain the residual of each node, which allows us to decide whether a node should be added to the `workqueue` or not based on its residual. Thus, in Algorithm 3, we can avoid the unnecessary repeated computations by filtering out the work in the `workqueue` in advance. In Section 7, we show that Algorithm 3 allows us to significantly reduce the number of arithmetic operations compared to Algorithm 2.

6 RELATED WORK

There exist many different types of link-based spamdexing techniques. Similar to the Anti-TrustRank, TrustRank [16] is also one of the well-known link-based spam detection methods where the idea is that normal pages tend to point to other normal pages. A method of propagating both trust and distrust values also has been considered [36]. Also, [15] measures the impact of link spamming on a page's rank while [2] proposes new classification features that combine link-based features and language model-based ones.

For the TrustRank-based [16] label propagation methods, [34] proposes to incorporate topical information of pages to select the seeds, and [35] suggests to expand the seeds by considering reputable seeds. Also, [1] proposes to use a graph-regularized classification to improve the accuracy of spam detection. For the ATR algorithm, PageRank-based seeds [16] and inverse PageRank-based seeds [18] are known to be useful. We compare our seeding method with these existing seeding strategies in Section 7. While most existing spam detection methods require a large set of labeled nodes, our seeding mechanism requires a very small portion of the node labels as described in Section 4.2. In this realistic setting, we observe that our seeding methodology is able to identify a high-quality set of seeds in the real-world web graphs.

A number of variations of PageRank have been proposed to accelerate PageRank computations including [21], [22], and [13]. In particular, [31] has proposed a multi-threaded data-driven PageRank algorithm which provides the motivation of our residual-based asynchronous ATR algorithm. Also, [22] recently proposed a method to approximate a pairwise personalized PageRank vector. We tried to customize the recent method proposed in [22] to our spam detection problem, and realized that our asynchronous ATR algorithm is much faster. While there has been extensive research on the variations of PageRank, it is meaningful to explore the convergence property of the asynchronous ATR algorithm with a specialized set for the spam detection task.

7 EXPERIMENTAL RESULTS

We conduct experiments on the real-world web graphs discussed in Section 2 to show the usefulness of the proposed methods.

7.1 Performance in Web Spam Detection

We test the performance of our seeding methodology for the ATR algorithm. Let ***QS*** (an abbreviation of Qualified Seeds) denote our seeding method described in Section 4.2.

7.1.1 Baseline Methods. To compare the performance in web spam detection, we consider three different classes of methods[4]: (i) classifier-based methods, (ii) TrustRank with different seeding methods, and (iii) Anti-TrustRank with different seeding methods.

For the classifier-based methods, we extract features for web pages, and train a classifier to classify the pages into normal or spam. To extract reasonable features for the pages, we try two different methods. In [26], various link-based features have been proposed to detect spam hosts. Also, some of our features presented in Table 4 are applicable to modeling a page. Thus, we extract these features for each web page and represent a page as a 10 dimensional feature vector. This baseline method is denoted by ***lfeat***. Also, we can compute node embedding features by using the node2vec method [14]. This baseline is denoted by ***nvec***.

We implement the TrustRank method [16] with a number of well-known seeding strategies. Among the seeding methods, we observe

[4]We also considered SpamRank [3] as a baseline, but we observe that its performance is not good enough, so it is not comparable to other baseline methods.

Table 6: The accuracy, F1 score, precision, and recall according to different numbers of examined pages. The *QS* method achieves the highest accuracy and F1 scores, and significantly outperforms other methods.

	No. of Examined Pages		*lfeat*	*nvec*	*trust*	*pr-page*	*ipr-page*	*pr-site*	*ipr-site*	*QS*
W1	4,000 (0.47% examined)	Accuracy	60.80%	94.50%	26.33%	96.00%	94.73%	96.41%	96.25%	**98.22%**
		F1 score	15.90%	5.80%	13.04%	45.67%	11.22%	53.90%	49.95%	**81.52%**
		Precision	9.00%	68.30%	6.98%	95.56%	98.43%	96.14%	99.05%	97.67%
		Recall	66.20%	3.00%	99.83%	30.01%	5.95%	37.45%	33.39%	69.95%
	6,000 (0.70% examined)	Accuracy	89.20%	94.60%	27.39%	96.12%	94.75%	96.86%	96.31%	**98.71%**
		F1 score	21.40%	22.10%	13.21%	48.20%	11.75%	61.98%	51.03%	**87.22%**
		Precision	18.00%	57.00%	7.07%	95.51%	98.27%	96.34%	99.01%	97.60%
		Recall	26.40%	13.70%	99.87%	32.23%	6.25%	45.69%	34.37%	78.83%
	10,000 (1.17% examined)	Accuracy	84.30%	94.40%	35.02%	96.21%	94.78%	97.47%	96.58%	**98.88%**
		F1 score	21.70%	30.90%	14.53%	50.16%	12.77%	71.46%	56.28%	**89.12%**
		Precision	15.10%	49.40%	7.84%	95.14%	98.09%	96.75%	98.89%	97.42%
		Recall	38.80%	22.50%	99.83%	34.06%	6.83%	56.65%	39.33%	82.13%
W2	4,000 (0.46% examined)	Accuracy	90.70%	92.50%	28.82%	94.79%	92.76%	95.56%	92.80%	**96.39%**
		F1 score	17.80%	4.10%	17.35%	48.21%	8.31 %	64.10%	9.32%	**69.07%**
		Precision	26.80%	73.10%	9.50%	97.10%	98.44%	97.93%	99.81%	98.19%
		Recall	13.30%	2.10%	99.92%	32.06%	4.34%	47.64%	4.89%	53.27%
	6,000 (0.69% examined)	Accuracy	90.50%	92.70%	30.11%	94.90%	92.77%	94.46%	92.88%	**98.12%**
		F1 score	5.00%	17.20%	17.62%	49.97%	8.72%	69.89%	11.20%	**86.03%**
		Precision	10.30%	60.60%	9.66%	96.95%	98.28%	97.87%	99.82%	98.43%
		Recall	3.30%	10.00%	99.90%	33.66%	4.56%	54.36%	5.93%	76.41%
	10,000 (1.14% examined)	Accuracy	73.90%	92.80%	37.51%	94.99%	92.80%	97.11%	94.01%	**98.60%**
		F1 score	28.60%	27.40%	19.30%	51.43%	9.53%	76.75%	34.64%	**89.95%**
		Precision	18.00%	58.30%	10.68%	96.77%	98.11%	98.01%	99.28%	98.22%
		Recall	68.90%	17.90%	99.88%	35.02%	5.01%	63.07%	20.98%	82.96%

		Prediction	
		normal	spam
Ground-truth	normal	a	b
	spam	c	d

$$\text{Accuracy} = \frac{a+d}{a+b+c+d}$$
$$\text{Precision } (p) = \frac{d}{b+d}$$
$$\text{Recall } (r) = \frac{d}{c+d}$$
$$\text{F1 score} = \frac{2rp}{r+p}$$
$$\text{No. of detected spams} = d$$

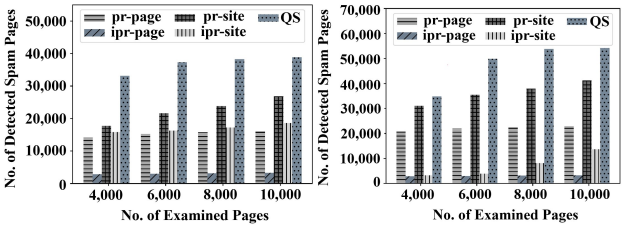
Figure 3: Evaluation metrics

that the PageRank-based seeds show the best performance where $\mathcal{L}$ is constructed by taking top-ranked pages according to their PageRank. Thus, we decide to use these seeds for the TrustRank method. This baseline is denoted by ***trust***.

For the ATR-based spam detection methods, we consider four different seeding methods. First, the page-level PageRank-based seeding is denoted by ***pr-page***. Second, as suggested in [18], we consider inverse PageRank (i.e., compute PageRank on the reverse page-level graph) to rank the pages. Let ***ipr-page*** denote this method. We can also apply these ideas to the site-level graph H to construct the site-level seed set. Since spam sites usually generate a lot of links around them (e.g., link farm described in Section 3.2), PageRank scores on H might allow us to select reasonably good site-level seeds. Let ***pr-site*** and ***ipr-site*** denote the site-level PageRank and inverse PageRank-based seeds, respectively.

7.1.2 Setting and Metrics. We notice that the performance of most existing spam detection methods had been tested based on the assumption that they are able to know the labels of a large portion of the nodes in a graph [11]. For example, [3] uses a 10-fold cross-validation where they assume that they know 90% of the node labels. However, in a real-world situation, it might be impractical to know that much portion of the node labels since labeling requires human efforts. Thus, in our experiments, we assume that we are able to examine only a small portion of the nodes by the site-level examination, e.g., about 10% of the nodes at the site-level graph. In this realistic setting, we observe that there is a large set of nodes whose ATR scores are zeros. Therefore, we do not need to set a threshold to determine whether a node should be classified into spam or normal. Instead, we consider a node to be spam if its ATR score is non-zero. This simple strategy allows us to correctly predict spam nodes. For the ATR-based methods, we run the residual-based asynchronous ATR algorithm with the tolerance $\epsilon = 10^{-8}$. Note that in our datasets (Table 1), around 90% nodes are normal meaning that if a spam detection method decides to predict all the nodes as normal nodes, the accuracy of the method is 90%. Thus, we focus more on the number of correctly detected spam pages and the F1 score for performance evaluation. Figure 3 shows the metrics we used to quantify the performance of each method.

7.1.3 Results. We assume that we are able to examine from 4,000 to 10,000 pages by the site-level examination. This corresponds to around 0.46% to 1.17% of the pages in our graphs. Table 6 and



(a) W1 dataset (b) W2 dataset

Figure 4: No. of Detected Spam Pages of the ATR Algorithm with Different Seeding Methods

Table 7: Performance of *sync*, *async*, and *rasync* on W1

		sync	*async*	*rasync*
$\epsilon = 10^{-4}$, $e = 4000$	No. of Detected Spam Pages	33,088	33,029	33,029
	F1 Score	81.52 %	81.67 %	81.67 %
	No. of ATR updates	51,384,240	46,680	46,454
	No. of Arithmetics	578,549,460	11,170,087	1,765,129
	Run Time (milliseconds)	7,596	339	87
$\epsilon = 10^{-8}$, $e = 4000$	No. of Detected Spam Pages	33,088	33,088	33,088
	F1 Score	81.52 %	81.52 %	81.52 %
	No. of ATR updates	100,199,268	83,961	83,972
	No. of Arithmetics	1,128,171,447	13,009,448	2,673,169
	Run Time (milliseconds)	14,952	358	99
$\epsilon = 10^{-4}$, $e = 10000$	No. of Detected Spam Pages	38,848	38,797	38,797
	F1 Score	89.12 %	89.31 %	89.31 %
	No. of ATR updates	52,240,644	54,327	54,093
	No. of Arithmetics	588,654,453	11,412,327	1,895,760
	Run Time (milliseconds)	7,678	350	98
$\epsilon = 10^{-8}$, $e = 10000$	No. of Detected Spam Pages	38,848	38,848	38,848
	F1 Score	89.12 %	89.12 %	89.12 %
	No. of ATR updates	101,055,672	95,631	95,634
	No. of Arithmetics	1,138,708,614	13,468,669	2,874,621
	Run Time (milliseconds)	14,628	374	111

Figure 4 show the results. Apparently, a good method should have a balanced precision and recall. Overall, the ATR-based methods show better performance than the classifier-based methods (***lfeat*** and ***nvec***) and the TrustRank method (***trust***). We note the TrustRank scores of most of the nodes are remained as zeros, and thus ***trust*** predicts almost all the nodes to be spam which leads to a high recall but a low precision. On the other hand, some of the ATR-based methods achieve better F1 scores meaning that given the same number of seeds, the ATR method might be able to more efficiently detect the spam pages than the TrustRank method. More importantly, we see that the ***QS*** method is able to achieve the highest accuracy and F1 scores on both W1 and W2 datasets, and the gap between ***QS*** and the second best method is significant. Figure 4 shows the number of detected spam pages of the ATR algorithm with different seeding methods. We see that ***QS*** is able to detect the largest number of spam pages.

7.2 Computational Cost of Synchronous and Asynchronous Anti-TrustRank

We investigate the computational cost of the synchronous and the asynchronous ATR algorithms. Let ***sync*** denote Algorithm 1, ***async*** denote Algorithm 2, and ***rasync*** denote Algorithm 3. We use the ***QS*** seeding strategy. By varying the tolerance value ϵ and the number of examined pages e, we compare the the number of detected spam pages as well as the F1 score. Also, we count the number of ATR updates and the number of arithmetic operations, and measure the run time. Table 7 shows the results on the W1 dataset. We get similar results on the W2 dataset.

We notice that when we set $\epsilon = 10^{-8}$, all the three methods return the identical results in terms of detecting spam pages. If we set $\epsilon = 10^{-4}$, there is a negligible difference between the synchronous and the asynchronous methods. More importantly, when we compare the number of ATR updates, the asynchronous algorithms, ***async*** and ***rasync***, make much fewer ATR updates than the synchronous algorithm, ***sync***. This is because the asynchronous algorithms maintain the `workqueue` to selectively process the nodes whose ATR scores need to be updated while ***sync*** processes all the nodes at every iteration.

In terms of the space complexity, the asynchronous algorithms require slightly more spaces than ***sync*** due to the `workqueue`. However, we note that the maximum length of `workqueue` is only around 4% of the number of nodes.

When we compare the number of arithmetic operations, the asynchronous algorithms also save much computation compared to ***sync***. We see that ***rasync*** significantly reduces the number of arithmetic operations compared to ***async***. As described in Section 5.3, ***rasync*** is able to efficiently reduce the size of the `workqueue` by exploiting the problem structure, which results in filtering out unnecessary computations.

We also compare our ATR algorithms with solution procedures designed for a general PageRank problem [13] [12]. We consider BiCGSTAB [13] denoted by ***bstab*** and the boundary restricted personalized PageRank algorithm proposed in [12] denoted by ***brppr*** in Table 8. For a fair comparison, we verify that ***bstab*** and ***brppr*** achieve the similar F1 scores with our method. In Table 8, we see that ***rasync*** is the fastest method.

Finally, we also implement the three algorithms, ***sync***, ***async***, and ***rasync*** using Spark 2.3.0 in a distributed system with 64 machines where each machine has four cores and 28G memory. Table 9 shows the results on large datasets. We note that ***rasync*** is also the fastest method in the distributed memory machines.

Table 8: Run Time (milliseconds) on W1 and W2

		sync	*async*	*rasync*	*bstab*	*brppr*
W1	e=4,000, $\epsilon=10^{-4}$	7,596	339	87	566	678
	e=4,000, $\epsilon=10^{-8}$	14,952	358	99	1,217	680
	e=10,000, $\epsilon=10^{-4}$	7,678	350	98	678	822
	e=10,000, $\epsilon=10^{-8}$	14,628	374	111	1,775	829
W2	e=4000, $\epsilon=10^{-4}$	6,526	556	148	821	726
	e=4,000, $\epsilon=10^{-8}$	13,841	1,205	374	1,926	742
	e=10,000, $\epsilon=10^{-4}$	6,212	607	169	707	968
	e=10,000, $\epsilon=10^{-8}$	13,174	1,406	453	1,546	948

Table 9: Parallel *sync*, *async*, and *rasync* on Distributed Machines

Data Information			Run Time (minutes)		
No. of nodes	No. of edges	Size of $\mathcal{S}$	*sync*	*async*	*rasync*
59,180,800	82,824,237	2,340,940	86	94	37
152,595,632	274,392,463	3,329,026	191	162	69
57,135,532	732,008,321	4,381,555	516	351	121
556,047,762	1,207,335,482	5,016,499	>2,116	>1,413	163

8 CONCLUSION & DISCUSSION

We develop a site-level seeding methodology for the ATR algorithm, which leads to remarkably boosting up the performance of the ATR algorithm. Also, we design a work-efficient asynchronous ATR algorithm which significantly reduces the computational cost of the traditional ATR method while guaranteeing convergence.

Among the label propagation-based spam detection methods, we focus on the ATR algorithm because the ATR method allows us to detect a large number of spam pages given a small set of high-quality seeds. While the TrustRank-based [16] methods require a large set of seeds to appropriately propagate the labels to a large portion of a graph, the ATR method can efficiently propagate the ATR scores to many spam pages from a small seed set if we select *core* spam pages as seeds due to the densely connected structure of link spam. Thus, the ATR method might be a reasonable and practical solution for a large-scale web spam detection problem.

Along with the seeding strategy proposed in this paper, other spam detection methods also can be used to locate good seeds for the ATR method. For example, we can make the seed set by including a set of top-ranked spam pages returned by content-based spam detection methods. Furthermore, the idea of the asynchronous ATR method can be easily extended to TrustRank [16]. Indeed, we can simultaneously consider TrustRank and Anti-TrustRank to increase the accuracy of spam detection. In this way, our methodologies can be integrated into other spam detection models [28] in practice.

We plan to develop an online ATR algorithm to handle evolving web graphs. Currently, we use a mini-batch approach to deal with constantly changing web graphs. We believe that we can incrementally update the ATR scores of a set of newly crawled pages. Also, we intend to improve our spam detection system by considering a more sophisticated analysis on the link structure [29] and also applying recently proposed graph embedding approaches [19].

ACKNOWLEDGMENTS

J. J. Whang is the corresponding author. This research was supported by NAVER Corp. and National Research Foundation of Korea funded by MSIT (2019R1C1C1008956, 2018R1A5A1059921).

REFERENCES

[1] J. Abernethy, O. Chapelle, and C. Castillo. 2010. Graph Regularization Methods for Web Spam Detection. *Journal of Machine Learning* 81 (2010), 207–225.
[2] L. Araujo and J. Martinez-Romo. 2010. Web Spam Detection: New Classification Features Based on Qualified Link Analysis and Language Models. *IEEE Transactions on Information Forensics and Security* 5, 3 (2010).
[3] L. Becchetti, C. Castillo, D. Donato, R. Baeza-Yates, and S. Leonardi. 2008. Link Analysis for Web Spam Detection. *ACM Transactions on the Web* 2, 1 (2008).
[4] A. Benczúr, C. Castillo, M. Erdélyi, Z. Gyöngyi, J. Masanes, and Michael Matthews. 2010. ECML/PKDD 2010 Discovery Challenge Data Set. https://dms.sztaki.hu/en/letoltes/ecmlpkdd-2010-discovery-challenge-data-set.
[5] M. Bendersky, W. B. Croft, and Y. Diao. 2011. Quality-biased Ranking of Web Documents. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. 95–104.
[6] L. Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (Oct. 2001), 5–32.
[7] 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).
[8] C. Castillo, D. Donato, L. Becchetti, and P. Boldi. 2007. WEBSPAM-UK2007. http://chato.cl/webspam/datasets/uk2007/.
[9] C. Castillo, D. Donato, A. Gionis, V. Murdock, and F. Silvestri. 2007. Know your Neighbors: Web Spam Detection using the Web Topology. In *Proceedings of the 30th International ACM SIGIR conference on Research and Development in Information Retrieval*. 423–430.
[10] Z. Cheng, B. Gao, C. Sun, Y. Jiang, and T. Liu. 2011. Let Web Spammers Expose Themselves. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*. 525–534.
[11] M. Erdelyi, A. Garzo, and A. Benczur. 2011. Web Spam Classification: a Few Features Worth More. In *Proceedings of the 2011 Joint WICOW/AIRWeb Workshop on Web Quality*. 27–34.
[12] D. Gleich and M. Polito. 2006. Approximating Personalized PageRank with Minimal Use of Web Graph Data. *Internet Mathematics* (2006).
[13] D. F. Gleich, L. Zhukov, and P. Berkhin. 2004. Fast Parallel PageRank: A Linear System Approach. *Yahoo! Research Labs Technical Report YRL-2004-038* (2004).
[14] A. Grover and J. Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[15] Z. Gyongyi, P. Berkhin, H. Garcia-Molina, and J. Pedersen. 2006. Link Spam Detection Based on Mass Estimation. In *Proceedings of the 32nd International Conference on Very Large Data Bases*. 439–450.
[16] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. 2004. Combating Web Spam with Trustrank. In *Proceedings of the 30th International Conference on Very Large Data Bases*. 576–587.
[17] J. Kleinberg. 1999. Authoritative Sources in a Hyperlinked Environment. *J. ACM* 46, 5 (1999), 604–632.
[18] V. Krishnan and R. Raj. 2006. Web Spam Detection with Anti-Trust Rank. In *Proceedings of the ACM SIGIR Workshop on Adversarial Information Retrieval on the Web*. 37–40.
[19] G. Lee, S. Kang, and J. J. Whang. 2019. Hyperlink Classification via Structured Graph Embedding. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1017–1020.
[20] J. Leskovec and C. Faloutsos. 2006. Sampling from Large Graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
[21] Q. Liu, Z. Li, J. Lui, and J. Cheng. 2016. PowerWalk: Scalable Personalized PageRank via Random Walks with Vertex-Centric Decomposition. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*.
[22] P. Lofgren, S. Banerjee, and A. Goel. 2016. Personalized PageRank Estimation and Search: A Bidirectional Approach. In *Proceedings of the 9th ACM International Conference on Web Search and Data Mining*. 163–172.
[23] F. McSherry. 2005. A Uniform Approach to Accelerated PageRank Computation. In *Proceedings of the 14th International Conference on World Wide Web*. 575–582.
[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[25] X. Qi and B. Davison. 2009. Web Page Classification: Features and Algorithms. *Comput. Surveys* 41, 2 (2009), 12:1–12:31.
[26] R. Silva, A. Yamakami, and T. Almeida. 2012. An Analysis of Machine Learning Methods for Spam Host Detection. In *Proceedings of the 11th International Conference on Machine Learning and Applications*. 227–232.
[27] N. Spirin and J. Han. 2012. Survey on Web Spam Detection: Principles and Algorithms. *ACM SIGKDD Explorations Newsletter* 13, 2 (2012), 50–64.
[28] C. Wei, Y. Liu, M. Zhang, S. Ma, L. Ru, and K. Zhang. 2012. Fighting Against Web Spam: A Novel Propagation Method Based on Click-through Data. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 395–404.
[29] J. J. Whang, Y. Hou, D. F. Gleich, and I. S. Dhillon. 2019. Non-exhaustive, Overlapping Clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 41, 11 (2019), 2644–2659.
[30] J. J. Whang, Y. Jung, I. S. Dhillon, S. Kang, and J. Lee. 2018. Fast Asynchronous Anti-TrustRank for Web Spam Detection. In *ACM International Conference on Web Search and Data Mining Workshop on MIS2: Misinformation and Misbehavior Mining on the Web*.
[31] J. J. Whang, A. Lenharth, I. Dhillon, and K. Pingali. 2015. Scalable Data-driven PageRank: Algorithms, System Issues, and Lessons Learned. In *Proceedings of the 21st International European Conference on Parallel and Distributed Computing*. 438–450.
[32] J. J. Whang, X. Sui, and I. Dhillon. 2012. Scalable and Memory-Efficient Clustering of Large-Scale Social Networks. In *Proceedings of the 12th International Conference on Data Mining*. 705–714.
[33] B. Wu and B. Davison. 2005. Identifying Link Farm Spam Pages. In *Proceedings of the 14th International Conference on World Wide Web*.
[34] B. Wu, V. Goel, and B. Davision. 2006. Topical TrustRank: Using Topicality to Combat Web Spam. In *Proceedings of the 15th International Conference on World Wide Web*. 63–72.
[35] X. Zhang, B. Han, and W. Liang. 2009. Automatic Seed Set Expansion for Trust Propagation Based Anti-spamming Algorithms. In *Proceedings of the 11th International Workshop on Web Information and Data Management*.
[36] X. Zhang, Y. Wang, N. Mou, and W. Liang. 2014. Propagating Both Trust and Distrust with Target Differentiation for Combating Link-Based Web Spam. In *Proceedings of the 25th International Conference on Association for the Advancement of Artificial Intelligence*.