Using patterns in the behavior of the random surfer to detect Webspam beneficiaries

Thomas Largillier and Sylvain Peyronnet

Université de Paris-Sud, Laboratoire de Recherche en Informatique, Bâtiment 490, F-91405 Orsay Cedex, France

Abstract. In order to appear in a good position on a search engine's result list it is not enough to be relevant regarding the request. Someone also have to be "popular". This notion of popularity is calculated by the search engine and is related to links made to the webpage.
In order to artificially increase their popularity, webmasters sometimes use malicious techniques referred to as Webspam. It can take many forms and is in constant evolution, but Webspam usually consists of building a specific dedicated structure of spam pages around a given target page. It is really important for a search engine to address the issue of Webspam otherwise it won't be able to provide users with fair and reliable results. In this paper we propose a technique to identify webspam through the frequency language associated with random walks amongst those dedicated structures. We identify the language by calculating the frequency of appearance of k-grams on random walks launch from every node.

1 Introduction

Nowadays, the Web has grown so big[1] that users cannot afford the patience to go all over it, or even through a rather small part of it. Except for their favorites sites they have to (and they do!) use search engines that answer to billions of requests per day. Search engines are thus facing the issue of providing their users with good results as quick as they can, results being web pages taken from a huge index (billions of web pages) and under a tremendous load (billions of requests each day).

To ensure good results to a particular request, search engines mostly use a relevance metric for web pages and requests. Being relevant regarding a certain query does not ensure being in the first places of a search engine result list. To arbitrate between equally relevant pages, search engines have to use other metrics. One of them is popularity. Most search engines are using a popularity mechanism that is not content related. Indeed, with a popularity independent of the content of webpages, computational issues linked to ranking web pages are less prevalent. Thus, popularity often depends on the links a webpage receive from other web pages. The Google's PageRank algorithm [14] compute the

[1] `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`

popularity of all pages independently of the query while Kleinberg HITS algorithm [10] has a query dependant version of the popularity called the authority score.

The economic model sustaining most websites is such that webmasters want their sites to appear on the first places of a search engines regarding specific requests. Indeed, the income of a website is directly correlated to unique visitors a website receive. Since search engines are the major sources of visitors on the web, to attract as many visitors as possible one has to maximize its exposure on them. Being close to the top will redirect a huge amount of traffic if the targeted request is wisely chosen.

Artificially increasing the relevance towards a request without quickly using spamming techniques and being spotted by search engines is almost impossible. Plus spammers often want to boost a legitimate page so they don't need to manipulate its relevance. So spammers aim to increase their popularity to move to the top of the list. The most effective way to increase the popularity of a given webpage is to create a set of dull pages organised in a specific architecture whose goal is to boost the target page. This is a borderline technique, which is far beyond the guidelines of most search engines. Structures intending to maximize the pagerank of one specific page are well known (see for instance the paper [7]). Those structures can no longer be use efficiently since it is hard for them to avoid automatic detection. Webspammers then slightly modify those structure to increase their rank while avoiding automatic detection.

There is thus a weapon race between search engines and spammers. It is really important for the first ones to deal with Webspam since it can pollute their results, then without fair results they will loose the users' confidence and visits. Having less visitors search engines will then loose their income due to advertising and sponsored links. Fighting spam is then an economic necessity for the search engines. For spammers this is the same problem: their income is correlated to their exposure in the SERPs (Search Engines Results Pages). So each time a search engine adapt itself to lower the incidence of Webspam, they have to renew their techniques to stay one step ahead.

In this paper we present a method whose goal is to identify malicious structures amongst web pages. The intuition behind our method is that spammers use specific pages architecture to route the PageRank around the target page in order to maximise its score while avoiding automatic detection. Since the PageRank can be seen as related to the behaviour of a random surfer, it seems that using random walks to reproduce the behaviour of this random surfer we will be able to expose paths created by spammers in order to manipulate and increase their pagerank.

The main results of this paper are:

- A method based on strong statistical results (borrowed from [6]) that allows to identify malicious patterns in the random walks.
- A methodology that classify random walks in similar categories in order to identify spammers and pages benefiting from their manipulations.
- Strong experimental results showing the efficiency of our approach.

This paper is organised as follow, in section 2 we present some related work regarding Webspam detection and demotion. Section 3 introduces our method to identify pages benefiting from Webspam. In section 4 we detail our experiments to verify the validity of our approach and show our results before concluding in section 5.

2 Related Work

Since search engines are based on well known popularity metrics such as the PageRank [14], spammers are trying to artificially boost their web sites with respect to these metrics. At the same time, Webspam has been extensively studied since it is really important for search engines to be able to deal with it.

Gyongyi *et al.* address in [7] the problem of constructing structures whose goal is to maximize the pagerank of a single page while De Kerchove in [5] solves the problem of maximizing the pagerank of a set of pages. Both papers are exhibiting optimal structures for this purpose. However, since these structures have a very rigid architecture (and so are easy to detect) , spammers are constructing less efficient, but slightly different structures. As we will see in the next sections, our method performs well even on lightly modified spam structures. More recently the interest was concentrated on the evolution of Webspam and Chung *et al.* proposed a link spam study in [4] to see how it evolves through a series of Web snapshots.

With the apparition of Webspam, many techniques were developed to deal with its effects in order to ensure the user with a fair ranking. There are two basic type of methods to deal with Webspammers, the first one is detection, *i.e* identifying Webspam or pages benefiting from it and demotion whose objective is to void the effects of such techniques without explicitly identifying Webspam.

In the spirit of the first category, Ntoulas *et al.* propose to identify spam through content analysis. The method use several criteria presented in [13]. Unfortunately, this method does not scale up since the Web is growing too fast and studying every page in depth is way to costly. Other approaches to identify cheaters on the Web rely on the structure of the Web and not on the content of the web pages.

The paper [8] by Gyongyi *et al.* proposes a method whose goal is to identify link spam through the estimation of pagerank coming from spam pages for every node. It is first required to find a set of "good" pages and then run a biased PageRank to find the part of all pages' pagerank that come from "good" pages. The drawback of this method is that it requires a preprocessing human step where people label pages as spam or non-spam.

Benczur *et al.* (see their paper [2]) propose a fully automatic detection method for Webspam by observing the distribution of contributing pages to suspected pages. Those who appear to have a biased distribution are considered as spam by their method.

Authors of [9] (resp. [11]) propose to give a Trust (resp. AntiTrust) score to pages to fight Webspam. These methods oblige the user to find a set of pages

she trust (resp. distrust) and then use a propagation scheme *a la* PageRank. It is then tricky to find a good seed set that will cover the whole graph. The notion of TrustRank was then refined by Baoning *et al.* to add topicality in [15].

Andersen *et al.* in [1] propose to compute a Robust PageRank by first approximating the supporting set of each page *i.e.* the set of pages that contribute to its pagerank.

Last, we proposed in a previous paper a lightweight technique based on clustering to demote the effects of Webspam and compute a fair pagerank for each page. This result is presented in [12]. To achieve the goal, we first cluster the Web graph using locally computable features and then run a PageRank where intra-cluster contributions are not taken into account. This is clearly a demotion based approach, while in this paper we focus on the problem of detecting spam structures.

3 Our method

In this section we describe the method we designed in order to detect Webspammers beneficiaries. It is based on random walks launched from suspected nodes in the graph. We then analyse how the PageRank is driven through the neighbourhood of those suspected nodes. Since the Web graph is very large, it is of the utmost importance to develop lightweight techniques to detect or demote Webspam. Random walks seems the natural choice for that purpose since they only induce a constant additional cost. Indeed, the crawler of the search engine must already cover the Web graph. The intuition behind the method we present in this paper is that spammers use specific pages architecture to route the PageRank around the target page in order to maximise its score while avoiding automatic detection.

The PageRank (see the seminal paper of Page *et al.* [14]) simulates the behaviour of a random surfer. This random surfer has two possible choices when visiting a page. he can either follow a link chosen uniformly at random on the page or he can teleport himself to another page on the graph. Spammers can not influence the teleportation but they can drive around the random surfer so he will come back quickly. Since the PageRank of a given Web page is basically the probability of being on that Web page at any moment of a random walk, this procedure will boost the pagerank of this given page. Thus using random walks to reproduce the behaviour of the random surfer we will be able to expose paths created by spammers in order to manipulate the random surfer and increase their pagerank.

We want to identify patterns in the random walks. We need now to define the patterns we will look for and use a confirmed methodology to classify random walks in similar categories in order to identify spammers and pages benefiting from spam.

During our random walks we need to store information that can be used at a global level to identify patterns. Using nodes' id won't be sufficient because we won't be able to draw patterns from such specific information. We need

less personal information about nodes while exploring the graph. We chose to associate nodes with their distance from the starting node of the random walk. We limit the distance to a constant d meaning that all nodes i which distance to the starting node is such that $d_i > d$ then $d_i = d + 1$. This neighbourhood is called the d-neighbourhood of the node i. Thus if we consider the neighbourhood of distance at most d the language has $d + 2$ symbols $[0 \cdots d + 1]$. This step is illustrated in Fig. 1a. Using distances instead of nodes' id will allow us to draw pattern and regroup nodes with similar random walk. It will also help us to recognize structures independently from their size. If structures serve the same objective they will have similar walks even if the sizes differ.

The information we are interested in in our random walks is how the pagerank is driven through the different levels in the neighbourhood. Thus we focus on the n-grams in our random walks. We then use the so-called ustat_k vectors on our random walks.

For a word w over the alphabet A we can compute the vector $\text{ustat}_k(w)$ which is a vector of size $|A|^k$. $\forall p \in [0 \dots |A|^k - 1], \text{ustat}_k(w)[p]$ represents the number of appearances of the k-gram p in w divided by $|w| - k + 1$, the number of blocks of size k in the word w. This is thus the frequency of p as a k-gram of w (more details about the theory behind the ustat_k vectors can be found in [6]).

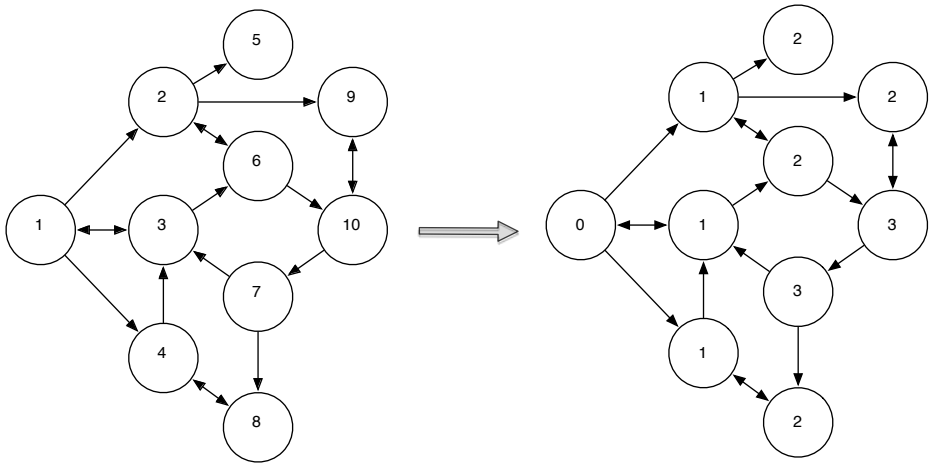
Using all these tools, we will now be able to identify similar structures, *i.e.* structures that produces mostly similar words. If two architectures produce similar words, it means that the PageRank is driven the same way around the target page (source of the random walk). If we can identify how a spammer route the PageRank in his neighbourhood we will be able to compare its ustat_k with the ones computed on suspicious nodes. The key point that make the use of ustat_k vectors highly effective for our goal is that they are robust, as the following result from [6] states. This proposition deals with the relation between L_1 distance, distance over words and ustat_k vectors.

Proposition 1. *For large enough words $w, w' \in \Sigma^*, \forall \delta > 0$, for large enough k:*

- *if $dist(w, w') \leq \delta^2$, then $||ustat_k(w) - ustat_k(w')||_1 \leq 7 \cdot \delta$.*
- *if $||ustat_k(w) - ustat_k(w')||_1 \leq \delta$ then $dist(w, w') \leq 7 \cdot \delta$.*

Thus if two ustat_k vectors over w and w' are close (in the L_1 sense) then w and w' are very similar.

We present in Fig. 2 the algorithm we derived from the previous proposition. This algorithm is used to match structures crawled amongst the Web graph against a library of previously known spam structures. It is correct thanks to the proposition. The figure shows the execution of the algorithm on a very small graph shown on the left of Fig. 1a. First in Fig. 1a, we label every node with their respective distance to the node 1 starting point of the random walk. Then we launch a random walk of size 16 resumed in Fig. 1b. The statistical projection of this random walk can be seen in Fig. 1c. The comparison with our pattern library will occur on the ustat_k vector.



(a) Step 1

$$\overline{0\ 1\ 2\ 3\ 3\ 2\ 1\ 1\ 0\ 1\ 2\ 3\ 3\ 1\ 0\ 1\ 2}$$

(b) Step 2

$$\text{ustat}_2^t = \big(0\ 3/16\ 0\ 0\ 1/8\ 1/16\ 3/16\ 0\ 0\ 1/16\ 0\ 1/8\ 0\ 1/16\ 1/16\ 1/8\big)$$

(c) Step 3

Fig. 1: Graphic description of our method

Our Algorithm: **Input** G the graph.
Input i the starting node.
Param d the neighbourhood distance.
Param l the length of the random walk.
Param k the size of the k-grams considered.

1. Compute the neighbourhood of distance d.
2. Launch a random walk of length l.
3. Compute the Ustat_k vector associated with the random walk.
4. Compare the vector with the library.

Fig. 2: Algorithm

We now look at the complexity of the algorithm. The first step is the computation of the neighbourhood of distance d for a node i and costs C_i which is defined as

$$C_i = 1 + \sum_{0 \leq k < d} \sum_{i \rightarrow_k j} d_i^+$$

where $i \rightarrow_k j$ means there exist a shortest path of length k between i and j and d_i^+ is the outdegree of node i. The worst case complexity scenario happens when a node i is able to reach all the nodes within its neighbourhood of distance d inducing a worst case complexity of $C_i = \mathcal{O}(n + m)$.

Summing this complexity for all nodes gives a running time

$$C = \sum_{i \in V} C_i = \mathcal{O}(n^2 + nm)$$

The complexity of launching a random walk of length l is constant for a node since it consists of $l - 1$ random choices. Random walks should be launched for all suspected nodes. Without previous knowledge on suspected nodes, it means it should be launched on all nodes with high pagerank or at most for every node in the graph. Thus this step requires $\mathcal{O}(n)$ steps.

The computation of the ustat_k vector for a random walk of length l requires also l steps since we need to scan the whole random walk. Then this step is in $\mathcal{O}(n)$ operations to compute the vectors for all nodes.

The complexity of this algorithm is the complexity of its first step meaning that it can run in time $\mathcal{O}(n^2 + nm)$. This mean that it is impossible to run the algorithm on the whole graph and that one should first select the node he wants to inspect, nodes in a certain range of pagerank for example.

4 Experiment

In this section, we present the experiments that have been conducted on the dataset WEBSPAM-UK2007[2]. This dataset is a crawl of the .uk domain made in May 2007. It is composed of 105 896 555 nodes. These nodes belong to 114 529 hosts and 6 478 of these hosts have been tagged. Please pay attention to the fact that hosts are tagged, not pages (e.g. entire domains instead of peculiar pages). We use the Webgraph [3] version of the dataset by Boldi and Vigna since it allows to manipulate huge graphs without using a lot of memory.

The hostnames are tagged with three different labels **spam**, **nonspam** and **undecided**. We are not interested in the last one since it does not provide discriminant enough information. In addition to those two sets we had a third set which we called **spam linked** since it is composed of nodes that are not in the spam set, but whose distance to nodes in the spam set is at most 2. Originally

[2] Yahoo! Research: "Web Spam Collections". http://barcelona.research.yahoo.net/webspam/datasets/ Crawled by the Laboratory of Web Algorithmics, University of Milan, http://law.dsi.unimi.it/. URLs retrieved 05 2007.

the spam linked set had 309 508 nodes but we remove from this set all the nodes that are also in the nonspam set. The intersection of those two sets contains 11 814 nodes. This is quite important and someone must keep in mind that they are probably many other suspicious nodes in the nonspam set. The final size of the spam linked set is then 297 694 nodes.

We then run a PageRank computation on the whole graph. We used the non-normalized version of the algorithm because of computer precision issues and run 60 iterations. The results are shown in table 1. We see in this table that, despite its few nodes, the spam linked set has and oddly high pagerank. This is surely explained by how we created this set. Many nodes have a huge part of their incoming links coming from the spam set. Thus it is surely in this set that we can find pages benefiting from Webspam. The latter probably more represented in the spam set. Indeed, this is a standard behavior for spammers to link from highly "spammy" Web pages their specific targeted (but clean) page.

	Nodes		PageRank	
	Value	Percentage	Value	Percentage
Graph	105 896 555	100	84 015 567	100
Spam	690 972	0.65	517 546	0.62
Spam Linked	297 694	**0.28**	7 733 663	**9.21**
Nonspam	5 314 671	5.02	4 230 292	5.04

Table 1: Number of nodes and pagerank part for all three sets

The next step of our experiments was to launched random walks from every node belonging to one of the three tagged sets. More precisely we launched one random walk for every set of parameters. Indeed we tried neighbourhood of distances 2 and 3 and looked at ustat_k vectors for bi- and tri-grams. The presented results are those obtained with a 3-neighbourhood and computing ustat_2 vectors. The size of the vectors is then 25.

	Number	Percentage
Spam	116 401	16.85
Spam Linked	16 497	5.54
Nonspam	609 307	11.46

(a) Sinks in each set.

	Number	Percentage
Spam	117 579	17.02
Spam Linked	47 654	16. 01
Nonspam	1 005 904	18.93

(b) Evasions in each set.

	Number	Percentage
Spam	8 406	1.22
Spam Linked	88 069	**29.58**
Nonspam	132 931	2.50

(c) Returns to the origin.

Table 2: General statistics.

First we want to look over each set for general statistics. More precisely, we are interested in knowing the proportion of **sinks** (nodes without outlinks) in each set as well as the number of random walks that leads to pagerank **evasion** *i.e.* the number of random walks that go to the $d+1$ (here 4) level more often than they come back. The results are shown in both tables 2a and 2b. We see that the set that minimizes both *criteria* is the spam linked set. Indeed sinks are clearly obstacles to a good circulation for the PageRank and evasion are not recommended if you want to maximise your pagerank. Note that those evasions may not all be real evasions since we do not make any difference between levels after 3.

In addition we look at random walks that come back to their origin at some point. People want to maximize their pagerank so they articulate the architecture around their target page to lure the random surfer. It is clear from results in table 2c that most cheaters are in the spam linked set. Pages in the spam set are pagerank accumulators and aggregators but don't do it for their own benefit.

Our technique proceeds by comparing the frequency vectors we compute to already known ones. Thus, we need a pattern library to compare the vectors to. Since we want to identify cheating structures, it is important to search for these patterns in the good set. We chose to extract the patterns we will look for in the spam linked set since it is supposed to contains people benefiting from spam.

As said previously, spammers needs to make the pagerank circulates but must ensure that this pagerank will come back really often. We then chose to extract all ustat_2 vectors, from the 100 nodes with the highest pagerank in the spam linked set, that present a high enough frequency of the pattern 01 meaning that the random walk came back to its starting point. We selected random walks that came back at least 5 times to their starting point. All selected patterns can be seen in table 4 in section 6.

This leads to fourteen patterns we will attempt to identify in all three sets. We consider that a vector v matches a pattern p whenever $|v-p|_1 \leq 0.2$. Note that the maximal distance between two vectors is 2.

	Number of matches	PageRank		
		Sum	Highest	Lowest
Spam	43	97.6	13.87	0.47
Spam Linked	**2460**	**107 372.51**	8704.22	0.48
Nonspam	1069	7002.65	517.55	0.36

Table 3: Results of comparisons

The results of these matches can be seen in table 3. We see in this table that they are very few matches for the spam set. This again provides evidence that nodes in this particular set are not here to benefit from Webspam, but are indeed Webspam and aim at producing artificial PageRank for nodes in the spam linked set.

Regarding the nonspam set. It has quite a few matches. It is reasonable to think that those matches are in fact cheaters since the intersection between the nonspam and the spam linked set is not null. We see that around 12% of those matches are in fact in the intersection.

It is in the spam linked set that we find the most matches that are really efficient. With only 2.3 times more matches than the nonspam set, the sum of the pagerank of the matched nodes is more than 15 times bigger.

It is interesting to note that pages whose random walks have a high number of returns to the origin mostly have a high pagerank. Since we can not massively find this pattern in the nonspam set, it can not be considered as a normal behaviour. These means that owners of these pages uses some kind of specific architectures in order to make the PageRank circulates around the page while maximizing the PageRank of the page itself.

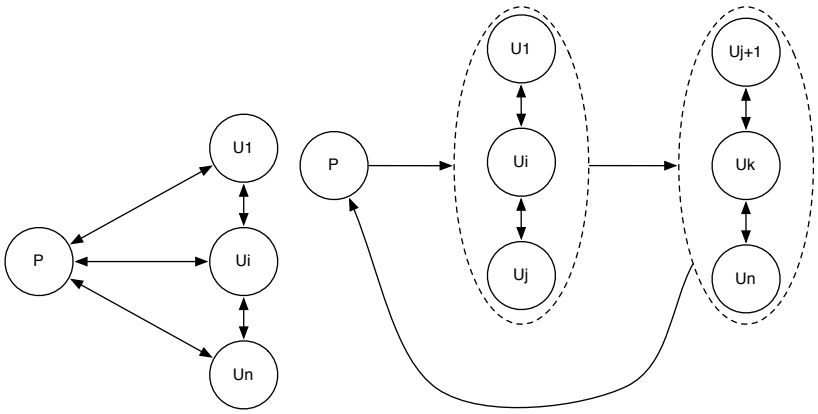
Looking at the patterns extracted in table 4 in the appendix, we can see that they can be divided in 2 categories presented in Fig 3. Of course those patterns are subjected to slight changes, loosing a bit of effectiveness but becoming really harder to identify.

In the first pattern (Fig 3a), we observe that the PageRank circulates a lot between the neighbors of the target page that exchanges a lot with its neighbors. This pattern may seem natural for a site but since it is rarely identified in the nonspam set, it is clear that only Webspammers set up this particular architecture to maximize their score.

The second pattern shown in Fig 3b creates short loops to bring the PageRank back to the target page. The PageRank is driven 2 steps away from the target page before coming back directly. This structure is rather usual in search engine optimization (SEO) since it is widely known by spammers that reciprocal linking is detected (and penalized) by the search engines, and so triangular linking is then set up to avoid detection.

Based on those promising results, it would be interesting to investigate further the effective patterns we can find in those ustat_k vector. We are able to find people benefiting directly from Webspam but spammers are getting smarter and their techniques more complicated every day. It could be also of interest to consider "middle men" in the Webspamming world to separate real cheaters from their pagerank providers. Those "middle men" are nodes taking part in smaller structures that boost their pagerank but not too much, then those pages can link the real target page that does not have to make reciprocal links and can avoid detection since it does not appear in a special architecture. The detection of those "middle men" is a lesser evil for spammers since it does not expose the real target page.

To understand in more depth the mechanisms built by spammers, ustat_k vectors for $k > 2$ should be analysed with more precision. Since they give more details about the spammers' strategy for PageRank redirection. They would also give a better mechanism for pattern comparison in terms of precision.



(a) First pattern (b) Second pattern

Fig. 3: Patterns extracted from spam linked top100

5 Discussion and Conclusion

In this paper, we presented a technique whose goal is to detect web pages that benefit from Webspam. This technique is built up strong theoretical foundations through the use of the ustat_k vectors of [6]. Experimental results show that the technique is both effective and efficient since we were able to detect cheaters using a few simple patterns. Moreover, our method is robust towards slight changes in the spam farms since it looks for small distances between ustat_k vectors. This means that we can also catch cheaters that use small modifications of the structure (*ie.*, the known pattern).

Constructing the pattern library is by itself an interesting problem, and can be efficiently done by observing already known spamming structures.

A drawback of our technique is that it may not be applied on the whole graph since the neighbourhood computation step may cost too much on graphs with high outdegree nodes. One should select first suspected nodes on several criteria: PageRank range, biased neighbors distribution, etc. Once the set of starting points is selected the method can be applied to detect efficiently the cheaters amongst the suspected pages.

Finally, we would like to emphasize on the assumptions we have assumed for this work. First, we are only considering link-based Webspam. Of course, it happens that some spammers are using totally legit structures but with spammy content. We are not dealing with the problem of detecting or demoting the impact of this kind of Webspam. Secondly, we assume that link-spam structures are different from natural structures. This hypothesis is correct as far as we are aware and is widely used in the literature.

References

1. Reid Andersen, Christian Borgs, Jennifer Chayes, John Hopcroft, Kamal Jain, Vahab Mirrokni, and Shanghua Teng. Robust pagerank and locally computable spam detection features. In *AIRWeb '08: Proceedings of the 4th international workshop on Adversarial information retrieval on the web*, pages 69–76, New York, NY, USA, 2008. ACM.
2. Andras A. Benczur, Karoly Csalogany, Tamas Sarlos, Mate Uher, and Máté Uher. Spamrank - fully automatic link spam detection. In *In Proceedings of the First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb*, 2005.
3. Paolo Boldi and Sebastiano Vigna. The webgraph framework I: Compression techniques. In *In Proc. of the Thirteenth International World Wide Web Conference*, pages 595–601. ACM Press, 2003.
4. Young-joo Chung, Masashi Toyoda, and Masaru Kitsuregawa. A study of link farm distribution and evolution using a time series of web snapshots. In *AIRWeb '09: Proceedings of the 5th International Workshop on Adversarial Information Retrieval on the Web*, pages 9–16, New York, NY, USA, 2009. ACM.
5. C. de Kerchove, L. Ninove, and P. Van Dooren. Maximizing PageRank via outlinks. *Linear Algebra and its Applications*, 429(5-6):1254–1276, 2008.
6. Eldar Fischer, Frederic Magniez, and Michel de Rougemont. Approximate satisfiability and equivalence. *Logic in Computer Science, Symposium on*, 0:421–430, 2006.
7. Z. Gyöngyi and H. Garcia-Molina. Web spam taxonomy. *Adversarial Information Retrieval on the Web*, 2005.
8. Zoltan Gyongyi, Pavel Berkhin, Hector Garcia-Molina, and Jan Pedersen. Link spam detection based on mass estimation. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 439–450. VLDB Endowment, 2006.
9. Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 576–587. VLDB Endowment, 2004.
10. Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.
11. Vijay Krishnan and Rashmi Raj. Web Spam Detection with Anti-Trust Rank. *AIRWeb 2006 Program*, page 37, 2006.
12. Thomas Largillier and Sylvain Peyronnet. Lightweight clustering methods for webspam demotion. In *In Proceedings of the Ninth international Conference on Web Intelligence*. IEEE Press, 2010.
13. Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. Detecting spam web pages through content analysis. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 83–92, New York, NY, USA, 2006. ACM.
14. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web, 1999.
15. Baoning Wu, Vinay Goel, and Brian D. Davison. Topical trustrank: using topicality to combat web spam. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 63–72, New York, NY, USA, 2006. ACM.

6 Appendix

00	01	10	11	12	20	21	22	23	30	31	32	33	34	40	41	42	43	44
0	1/8	1/48	1/24	17/48	1/12	9/48	1/48	1/12	0	1/16	1/48	0	0	0	0	0	0	0
1/24	7/48	1/8	7/12	1/24	0	1/24	1/48	0	0	0	0	0	0	0	0	0	0	0
1/48	7/48	5/48	13/24	1/12	1/48	1/16	1/48	0	0	0	0	0	0	0	0	0	0	0
0	7/48	1/48	1/8	1/6	1/48	0	5/48	1/6	1/12	1/48	1/24	0	1/48	0	1/48	0	0	1/16
1/48	1/6	0	1/8	5/24	1/16	1/48	1/48	7/48	1/16	1/48	1/48	0	1/24	1/48	0	1/48	0	1/24
0	7/48	5/48	23/48	1/16	1/48	1/24	5/48	1/48	0	0	1/48	0	0	0	0	0	0	0
1/48	1/8	1/24	7/16	7/48	1/16	1/12	1/48	1/48	0	0	1/48	1/48	0	0	0	0	0	0
0	5/24	3/16	5/48	5/24	0	1/8	0	1/12	0	1/12	0	0	0	0	0	0	0	0
0	1/4	1/48	0	11/48	3/16	0	1/16	1/24	1/48	0	0	3/16	0	0	0	0	0	0
1/48	5/16	1/48	1/48	7/24	1/4	0	0	1/24	1/24	0	0	0	0	0	0	0	0	0
0	1/8	1/12	13/48	5/48	1/48	1/48	0	1/16	0	1/48	0	1/24	1/12	1/48	1/48	0	1/24	1/12
1/12	7/48	1/12	1/8	1/12	0	0	0	1/12	0	0	0	0	1/12	1/24	1/24	0	0	11/48
5/48	5/24	3/16	1/8	1/12	0	0	0	1/16	0	0	0	0	1/16	0	1/16	0	0	5/48
0	7/48	0	1/48	5/12	1/8	7/24	0	0	0	0	0	0	0	0	0	0	0	0

Table 4: Patterns used for recognitions