Unsupervised Domain Ranking in Large-Scale Web Crawls

YI CUI, Texas A&M University, USA
CLINT SPARKMAN, United States Air Force Academy, USA
HSIN-TSANG LEE, Microsoft Corp., USA
DMITRI LOGUINOV, Texas A&M University, USA

With the proliferation of web spam and infinite autogenerated web content, large-scale web crawlers require low-complexity ranking methods to effectively budget their limited resources and allocate bandwidth to reputable sites. In this work, we assume crawls that produce frontiers orders of magnitude larger than RAM, where sorting of pending URLs is infeasible in real time. Under these constraints, the main objective is to quickly compute domain budgets and decide which of them can be massively crawled. Those ranked at the top of the list receive aggressive crawling allowances, while all other domains are visited at some small default rate. To shed light on Internet-wide spam avoidance, we study topology-based ranking algorithms on domain-level graphs from the two largest academic crawls: a 6.3B-page IRLbot dataset and a 1B-page ClueWeb09 exploration. We first propose a new methodology for comparing the various rankings and then show that in-degree BFS-based techniques decisively outperform classic PageRank-style methods, including TrustRank. However, since BFS requires several orders of magnitude higher overhead and is generally infeasible for real-time use, we propose a fast, accurate, and scalable estimation method called TSE that can achieve much better crawl prioritization in practice. It is especially beneficial in applications with limited hardware resources.

CCS Concepts: • **Information systems** → **Web crawling**; **Page and site ranking**;

Additional Key Words and Phrases: Web crawling, ranking, frontier prioritization

ACM Reference format:
Yi Cui, Clint Sparkman, Hsin-Tsang Lee, and Dmitri Loguinov. 2018. Unsupervised Domain Ranking in Large-Scale Web Crawls. *ACM Trans. Web* 12, 4, Article 26 (September 2018), 29 pages.
https://doi.org/10.1145/3182180

1 INTRODUCTION

Rapid growth of the World Wide Web (WWW) has had a dramatic impact on our society through numerous online innovations such as e-commerce, distributed file sharing, and social networking. Users often find information and services through *search engines*, which in turn rely on *web*

Authors' addresses: Y. Cui, Texas A&M University, Department of Computer Science and Engineering, College Station, TX, 77843; email: yi@cse.tamu.edu; C. Sparkman, United States Air Force Academy, 2354 Fairchild Drive, Suite 6G-101, Colorado Springs, CO, 80840; email: Clint.Sparkman@usafa.edu; H. T. Lee, Microsoft Corp. Redmond, WA, 98052; email: htlee@microsoft.com; D. Loguinov, Texas A&M University, Department of Computer Science and Engineering, College Station, TX, 77843; email: dmitri@cse.tamu.edu.

1559-1131/2018/09-ART26 $15.00
https://doi.org/10.1145/3182180

crawlers to discover valuable pages and maintain a current picture of the web. Competition for high placement in search engine results (both for legitimate and for malicious purposes) has created a financial incentive for many unethical Internet practices intended to deceive search engines and manipulate their ranking algorithms. In addition to adversely impacting the quality of commercial search results, web spam frequently impedes web exploration for various research purposes in areas of networking, data mining, information retrieval, and data-intensive computing. As a result of this adversarial nature of the web, crawlers should be designed to avoid "undesirable" content as much as possible. This requires using algorithms that possess not only sufficient accuracy but also manageable overhead to be executed in real time.

Traditional crawlers [13, 14, 21, 29, 30, 41, 42, 43, 52, 55, 56, 61, 62, 66, 68, 70] rely on BFS frontier prioritization. However, experience shows that BFS eventually encounters massive spam sites that use scripts to dynamically generate a huge number of pages and/or hostnames in an attempt to "trap" search engines and influence their ranking results [48]. Additionally, visiting malicious domains exposes crawlers to sabotage tactics (e.g., slow/incorrect DNS responses and dragged-out HTTP sessions), which can severely degrade performance and stall progress if not addressed. Given the virtually unlimited web space and impossibility of downloading every discovered URL, the main problem in crawler design boils down to *budgeting*—allocating limited resources (e.g., bandwidth, CPU cycles, storage) to exploration of valuable sites rather than spam. There are two distinct approaches for doing so, which we discuss next.

The first direction requires sorting the frontier by PageRank [7, 22] and crawling pages with the highest score in the pending queue first. While reasonable in theory, this technique has never been tested in wide-scale crawls due to its apparent computational infeasibility. For example, consider IRLbot's web graph [48] with 310B edges (i.e., 2.5TB), where periodic computation of 50 iterations of PageRank over graphs 156× larger than RAM (i.e., 16 GB) would be no simple task. In fact, even if the 510GB (i.e., 32× RAM) PageRank vector π could have been computed, maintaining the frontier with 35B pages (i.e., 3.8TB, or 240× RAM) in descending order of PageRank would be an even bigger challenge. This is in addition to regular IRLbot operation, which included managing 10K concurrently open connections, decompressing gzipped pages, parsing HTML at close to 1Gbps, checking URL uniqueness at 180K links/sec, enforcing politeness in multiple heaps, performing DNS lookups and robot checks, building a web graph, and saving detailed trace statistics, all with just four CPU cores. While simulations suggest [22] that PageRank frontier prioritization finds top-ranked pages quicker, other proposals have emerged to lower its computational cost. These methods use vector π based on such metrics as page in-degree [22], a single iteration of PageRank called OPIC [2], batch-mode PageRank [22], and the number of pending pages per site [15]. Even though they reduce the complexity of obtaining π, the cost of sorting the huge frontier, i.e., the main bottleneck, remains unchanged. The only effort to utilize page-level prioritization in non-trivial crawls over the Internet is ClueWeb09 [24], which scheduled the pending URL queue using OPIC. The experiment downloaded 1B pages in 2009 using the massive Google-IBM-NSF cluster that in 2008 contained 90K nodes [25]. On top of that, it was later reported [3] that the ClueWeb09 parser discarded all dynamic links from the frontier. As we discuss below, this reduced its web-graph size from 57B edges to 7.9B. In a research setting that cannot afford deployment of crawler code in highly parallelized clusters or removal of dynamic links, frontier prioritization is still a generally difficult task.

The second direction, proposed in IRLbot [48], suggests that crawlers perform *admission control* on queued URLs, where each domain is given a certain budget based on its perceived reputation and only URLs satisfying the budget are crawled within each time window. The main difference of this method was that it does not sort the pending queue, but instead operates using linear scans that shuffle URLs passing their budgets into a set of priority queues and those failing it into a

separate backlog file. The method has good scalability as crawl size increased and its performance is not affected by the amount of backlogged links. Other benefits to using domain graphs for real-time ranking include their significantly smaller size and better resilience against manipulation since hijacking links from a large number of legitimate domains is more costly than reproducing the same effect on the page-level graph. Domain budgeting may be the only suitable prioritization technique for large-scale crawlers operating without extravagant cluster hardware.

1.1 Ranking Domains

Spam websites that receive small budgets (e.g., 10 pages per day or per week) are generally not a threat to crawlers. They can be downloaded at the same rate as obscure personal pages and later passed on to various classifiers that will decide their usefulness for inclusion in the search index. On the other hand, highly branching spam presents a real problem as it clogs the crawler's queue and prevents expansion into other parts of the Internet. For example, a target that sends 1K links to random hostnames and uses a wildcard DNS domain can pollute the frontier with 1B pages after just three levels of BFS. In early versions of IRLbot that did not use prioritization, we encountered such cases regularly and failed to advance the crawl to a nontrivial depth from the seed page. Since domains with a large number of unique URLs may contain legitimate information (e.g., `ebay.com` with 40M pages or `facebook.com` with 1.5B), the main problem examined in this work is *how to differentiate between domains that should be massively crawled and those that should not.*[1] There are two performance metrics in achieving this classification—overhead and accuracy—where the former refers to the amount of processing needed to compute reputation and the latter describes the ability of the corresponding algorithm to avoid allocation of disproportionately many resources to low-quality domains.

Due to the massive size of the web and constantly evolving spam tactics, we only consider *unsupervised* methods for deciding domain reputation. In a commercial search engine setting, usage of human input (e.g., user clicks, training on manually identified spam) is perfectly acceptable; however, for a typical research crawler, these solutions are either impossible (i.e., due to lacking access to necessary data) or costly to maintain (i.e., due to the huge manual effort needed to keep up with the changing web). Thus, we consider a crawler that uses only the link structure of the seen web to dynamically compute domain reputation scores $\pi_1, \ldots, \pi_n$ using some prioritization algorithm $\mathcal{P}$, where n is the total number of known domains. Instead of directly using these scores, the crawler sorts set $\{\pi_i\}$ and obtains *reputation ranking* $r_1, \ldots, r_n$, where r_i is the position of domain i in the sorted list. Then, a budget function $f(r_i)$ decides the amount of bandwidth allocated to each domain according to its ranking position. To keep complexity low and avoid the need to differentiate between obscure sites, suppose that $f(x)$ monotonically decreases for the top R domains and remains a small constant for all $x > R$. Nodes at the top of the list get significantly larger budgets (e.g., by three to five orders of magnitude) that may be aggressively (i.e., nonlinearly) scaled as r_i gets smaller. As a result, the goal of the reputation algorithm is not to identify each spam page or detect malicious content, as done previously [9, 10, 12, 23, 27, 33, 37, 58, 65, 75, 76, 77, 78, 82], but rather to prevent domains with poor or unknown quality from admission into the top R list.

While [48] has shown that IRLbot's reputation rank was correlated with the number of pages pulled from each domain, which suggests that the budget enforcer was successful, there is no evidence that domains with high ranking in the final dataset were in fact overwhelmingly reputable. Therefore, our goal in this article is to understand the quality of top domains produced by IRLbot's

[1]Note that this problem is entirely different from search-related ranking of results, which involves fine-granular decisions about the quality of each page (e.g., based on content analysis).

in-degree ranking, compare it to alternative unsupervised classification mechanisms, and examine whether better methods can be designed while keeping the computational cost at similar levels.

1.2 Main Results

We start with manual analysis of four algorithms: PageRank [14], OPIC [2], In-Degree (IN) [48], and level 2 Supporters (SUPP) [12], where the first three have been proposed for crawl prioritization and the fourth one is new. Our investigation is based on the two largest academic datasets, i.e., a 6.3B-page IRLbot crawl [48] and a 1B-page ClueWeb09 experiment [24]. To the best of our knowledge, no prior work has studied the domain graphs in these datasets or performed ranking over them. For each algorithm, we open candidate domains in a web browser and classify them as spam or nonspam. Using $R = 1{,}000$, we find that both PageRank and OPIC produce significantly worse rankings than the other two methods, admitting 30+ spam domains into the top R list and exhibiting low-quality domains throughout the entire ranking range $[1, R]$. In both cases, the highest-ranked spam domain appears in the top 10 list, which in practice may lead to devastating results depending on how $f(x)$ allocates its budgets. We also discover that these two techniques exhibit similar ranking, despite PageRank's substantially higher overhead. IN reduces the amount of spam 3 to 4× in the top R list, with its first spam domain appearing in position 25 in IRLbot and 83 in ClueWeb09. Its ranking consists almost entirely of well-known sites, which is surprising given its simplicity and low cost. SUPP, being a generalization of IN to two hops, performs even better, admitting only one spam domain in IRLbot and two in ClueWeb09.

In general, manual comparison of web-ranking techniques is not only a labor-intensive but also a subjective task since there is no commonly agreed-upon standard by which one can easily identify spam. We therefore obtain additional indicators from the Internet to help measure the relative value of the domains in question. The first method relies on Google Toolbar Ranks (GTRs) [34], which are integers in the range of 0 to 10, widely believed to be related to the importance of the corresponding page. Certain URLs do not have a GTR, which puts them into a separate category. Our manual examination of top 1,000 lists of different ranking methods finds that 43% of those with GTR = 0 and 32% of those without a GTR are spam. The same probability for GTRs higher than 4 is almost zero. The second technique utilizes public spam lists to understand how many blacklisted domains are ranked near the top by each method. Results show that SUPP decisively outperforms the other three algorithms in every comparison and often admits 10× fewer undesirable domains into its top R list, where R is expanded to 10K for GTRs and 100K for the blacklist.

To create a competitive alternative to SUPP, we next investigate the possibility of using another classical technique, TrustRank [40], in an unsupervised setting. Normally, TrustRank requires selection of a reputable whitelist to which all teleportation takes place. To avoid manual input, we investigate several techniques for autogenerating the whitelist and select the best strategy to face off against SUPP. Analysis shows that TrustRank easily beats PageRank/OPIC; however, it struggles to improve IN in the majority of comparisons and is much worse than SUPP in all cases. Although SUPP is without a doubt superior in our evaluation, it requires an enormous amount of CPU processing due to elimination of duplicates in BFS and a large number of random memory accesses (e.g., 5 trillion in the IRLbot case). This makes it difficult to incorporate into a high-performance web crawler. To overcome this problem, we propose a novel technique to approximate SUPP, which we call *Top Supporters Estimation (TSE)*. We compare it against the exact SUPP algorithm and the Bit Vector (BV) estimation method from [10]. Our results show that TSE achieves less than 1% average error in the top R list, while reducing the runtime of SUPP by 36,000×. It is also 10× faster than BV, with 6× lower error. For external-memory application of TSE when the graph does not fit in RAM, we show that its disk I/O is 1,000× less than SUPP's and its RAM requirement is

4,000× smaller. TSE additionally reduces BV's RAM requirement by three orders of magnitude, all of which makes it highly appealing for real-time use in crawlers with limited resources.

2 RELATED WORK AND DISCUSSION

2.1 Web Crawlers

Much academic and corporate effort has gone into designing web crawlers [13, 14, 21, 24, 30, 44, 46, 52, 55, 56, 61, 66, 84]. University endeavors typically download 100M to 150M pages [21, 43, 47, 66, 71, 83]; however, due to the small scale of these crawls, frontier prioritization has not received much attention during actual experiments. While ClueWeb09 [24] reached 1B pages using OPIC, this required a massive Google-IBM-NSF cluster and removal of dynamic links, as mentioned earlier. Industry labs report 1B to 5B pages [36, 55] and commercial search engines were observed in 2008 to index 30B to 40B pages [48], with links to over 1T unique URLs [5]. In late 2010, Yahoo disclosed crawling 150B pages and using graphs with 5T edges [60]. In 2012, Google was crawling 20B pages/day and keeping track of 30T unique URLs [67]. However, not much is known about their prioritization schemes, the size of server clusters involved in crawling, and the cost of maintaining the classification engine that guides their crawlers.

Our previous effort [48] has led to a high-performance crawler called IRLbot that can perform multi-billion-page web exploration using a single server, which is accomplished using several novel algorithms for verification of URL uniqueness, reputation ranking, and enforcement of budgets. IRLbot uses domain in-degree to decide reputation, where the budget function $f(x)$ is linear in the top 10K domains and constant otherwise. See [48] for an overview of its algorithms and [3] for the statistics of its 6.3B-page crawl.

2.2 Spam Detection and Avoidance

Ranking algorithms can be broadly classified as either *static* or *dynamic*. The former category deals with assigning rank to each resource on the web (e.g., page, host) using query-independent techniques, while the latter determines the best documents among only those relevant to a particular set of keywords. Crawlers deal solely with static ranking, which can be further partitioned into *supervised*, which relies on training data and human input that give hints as to what constitutes desirable (whitelisted) or undesirable (blacklisted) content, and *unsupervised*, which operates using information only directly measurable by the crawler.

A significant bulk of previous work lies in the supervised category, which uses such indicators as web topology [9, 10, 11, 16, 17, 40, 37, 57, 77, 78], page content [1, 27, 33, 45, 53, 58, 63, 75, 76, 82], and user clicks/browsing habits [6, 8, 51, 64, 72]. However, the main challenge with these methods in a research setting is the cost involved in maintaining up-to-date training data, scaling a handful of spam examples to billions of pages, and acquiring human click traces. For example, the commonly used Webspam-UK2007 training database has 222 hosts labeled as "spam" and 277 as "undecided," in a collection of 114K sites. It is unclear how much these 499 suspicious entities help in classifying content in larger datasets, such as IRLbot's with 641M hosts, and whether the same type of spam is found in crawls outside of .uk or after 2007. User clicks provide great information; however, they cannot be obtained at sufficient scale outside of commercial search engines with billions of customers.

For unsupervised methods, it is important to further differentiate between *spam detection* and *spam avoidance*. The former techniques typically seek to identify bad nodes in a given dataset. This direction is exemplified by [23, 65], which decompose the web graph into strongly connected components to find spam farms. While appropriate in certain scenarios, spam detection does not help with scalable frontier prioritization, where crawlers need to determine only the top websites

that should be massively crawled. In fact, differentiating between spam and obscure legitimate pages beyond the top R list is unnecessary since both can be crawled at some slow default rate. On the other hand, spam avoidance tries to determine the best nodes in a given graph and generally requires assigning numerical scores to *all* nodes, i.e., ranking them.

To achieve reasonable complexity and independence of manual input, we consider algorithms that extract reputation based on endorsement from other nodes in the topology constructed by the crawler, i.e., incoming link structure at a certain depth from each node. Assume an n-node directed unweighted graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Then, let $d(i, j)$ be the *shortest* distance between i and j along the directed edges in G. We say that i provides level D support to j if $d(i, j) = D$. We then compute the reputation score of each node j using an iterative process

$$\pi_j^t = \sum_{d(i,j)=D} h_i\left(\pi_i^{t-1}\right) + \epsilon_j, \tag{1}$$

where t is the iteration number, $h_i(.)$ is some function of node i's current weight, and ϵ_j is an additive constant that may depend on the target node j.

For $D = 1$, perhaps the simplest form of Equation (1) uses $h_i(x) = 1$, $\epsilon_j = 0$, and one iteration, which produces the well-known *In-Degree* (IN) [48–50]. Assuming X_j is a random variable that models the in-degree of node j, the complexity of IN is $nE[X_j]$, which is linear in n if the average degree stays constant as $n \to \infty$. The second family of methods falls under the classic PageRank algorithm [14], which has served as the foundation for numerous other approaches such as SiteRank [80], HostRank [31], ParentPenalty [78], SpamRank [12], AggregateRank [32], and Truncated PageRank [10], to name a few. PageRank models a random walk on G, where the walker either traverses one of the outgoing edges with probability α (usually set to 0.85) or teleports to a random node in the graph with probability $1 - \alpha$. The PageRank score π_j for page $j \in V$ is the stationary probability of being visited by the random walker, which is the limit to Equation (1) as $t \to \infty$ using

$$h_i(x) = \alpha \frac{x}{Y_i}, \quad \epsilon_j = (1 - \alpha)\frac{1}{n}, \tag{2}$$

where Y_i is the out-degree of node i. After convergence of Equation (1), where 40 to 50 iterations are usually sufficient in practice, all nodes are sorted in descending order of their π_j and the ranks are assigned sequentially from 1 to R. As expected, PageRank is more computationally expensive than IN as it requires multiple passes through the graph. In cases when G does not fit in RAM, this may place a substantial burden on the crawler. To reduce complexity, a single iteration of PageRank can be used with $\alpha = 1$ and $\pi_i^0 = 1$, which is known as *Online Page Importance Computation* (OPIC) [2] or *Weighted In-Degree* (WIN) [20]. Its rationale is to assume that each node i in the graph has one unit of "authority" that it distributes equally between the nodes it links to. Compared to IN, this method aims to curb the amount of influence any node can have on the overall ranking within the system.

For $D = 2$, the only technique used in the literature is *Supporters* (SUPP) [10], which uses BFS along the in-graph to count the number of unique nodes reachable at distance 2 from each node. Elimination of duplicates during this process is a computationally expensive proposition, especially when G does not fit in RAM or the graph is highly branching. Complexity of SUPP is $nE[X_i Y_i]$, which can be orders of magnitude larger than $nE[X_i]E[Y_i]$ due to correlation between in/out-degree. On the bright side, usage of depth-2 supporters makes acquisition of large π_j more difficult for spammers than manipulating simpler techniques, such as IN. To our knowledge, no prior paper has computed exact SUPP on graphs of this scale or attempted to use it for ranking as a standalone method (e.g., [10] uses approximate SUPP counts to help detect spam).

2.3 Domain Ranking

The main idea behind IRLbot's ranking strategy is to attach financial responsibility to each node that provides support to j, which requires constructing a graph between nodes that someone has to register manually, typically through a payment. In its current state, DNS does not define mechanisms that can easily differentiate domains from hosts or allow one to quickly determine what parts of the Internet are under control of a single entity. While domain graphs have been engaged in other contexts (e.g., for improving page-level ranking [81]), there was no algorithm for systematic parsing of domains out of URLs. Therefore, IRLbot introduced the concept of *Pay-Level Domains* (PLDs) [48], which are DNS names that must be purchased at a gTLD or cc-TLD registrar. In some cases, they are one level below a gTLD (e.g., google.com) or two levels below a cc-TLD (e.g., amazon.co.uk); however, there are numerous exceptions (e.g., riverlands.wa.edu.au). IRLbot used a custom list of 1,761 domain suffixes, which we constructed by examining the registration structure of each available TLD at that time. Since then, the TLD/cc-TLD structure of DNS has grown significantly; however, there exists a Mozilla project [54] that maintains a current set of PLD suffixes (8,001 as of October 2017). This makes it easy to map URLs to their PLDs in future crawls and keep the rules current.

PLD graphs are constructed from the corresponding web graph by condensing all pages contained within a domain into a single node and discarding duplicate edges. They are arguably less vulnerable to manipulation than page-level or site-level constructs. Indeed, to achieve high PLD-level ranking, spammers are forced to either attract links from a diverse set of legitimate PLDs or purchase/maintain many domains of their own. Furthermore, PLD graphs are dramatically smaller in size than alternative data structures used in prior work [12, 16, 31, 32, 40, 51, 80], which enables much more efficient ranking during large crawls. The main limitation of domain-based budget enforcement is that all pages in a PLD are treated equally by the crawler. It is currently unknown if this presents a problem, but it is conceivable that the same admission-control idea can be applied to hosts within each domain and even directories within each host, as long as the ranking algorithm is sound. Therefore, evaluation in this article is the first step to understanding how well these techniques can function in practice.

3 MANUAL ANALYSIS

3.1 Crawler Objectives

Controlling the reputation of downloaded content not only reduces bandwidth waste, as discussed in the introduction, but also curtails the influence of unreputable pages on the collected dataset. Suppose G_∞ is the infinite web graph and $G_\mathcal{P} \subset G_\infty$ is its representation captured by a crawler with prioritization function $\mathcal{P}$. If $G_\mathcal{P}$ has a significant bias toward domains with undesirable content, it is possible that the search engine may have a hard time discerning good pages from bad in $G_\mathcal{P}$. For example, PageRank [14] teleports randomly to each page with probability $1/n$. Therefore, the more spam pages the crawler has visited, the larger their combined teleportation mass and the more the support to their target pages. In fact, a spammer may not need to attract many external links if the crawler covers its spam farm with sufficient diligence.

By altering the collected graph using a prioritization scheme $\mathcal{P}$, the crawler can reduce the weight of undesirable pages *preemptively* and provide a much cleaner input to the ranking algorithm used later by the search engine. One option for achieving this is to aim for $G_\mathcal{P}$ to be *top-rank equivalent* to G_∞ (i.e., top nodes in G_∞ are top in $G_\mathcal{P}$ according to some ranking function and vice versa), but this leads to problems since G_∞ is likely dominated by the infinite mass of autogenerated content (e.g., spam farms, scripted URLs), usually of questionable nature. Instead, define $G_U \subset G_\infty$ to be a subgraph induced by only "useful" pages and their links in G_∞. Then, the crawler's ultimate

objective might be achieving top-rank equivalence to G_U using its prioritization function $\mathcal{P}$. However, since G_U is unknown and generally a subjective entity, a more realistic goal is to ensure that top-ranked nodes in $G_{\mathcal{P}}$ are a *subset* of the top-ranked nodes in G_U. The remaining (low-ranked) nodes are of little concern as they are allocated small default budgets, which do not allow them to significantly influence the direction of the crawl or the structure of $G_{\mathcal{P}}$. However, very little evidence exists about what nodes dominate $G_{\mathcal{P}}$ among the existing crawlers [13, 14, 21, 30, 41, 43, 52, 55, 61, 62, 66, 68, 70], and no evaluation standard exists. Therefore, we first design a framework for understanding what domains are most prominent in $G_{\mathcal{P}}$ and later examine whether IRLbot's ranking function can be improved.

3.2 Methodology

Deciding the quality of a web-ranking algorithm is challenging for two reasons. The first problem is the absence of common methodology by which to measure ranking results in spam avoidance applications. Related work [12, 16, 40, 79] usually selects a small random sample of the graph and manually classifies it to determine if each page is good or spam. Then, the entire ranking list is split into buckets and the algorithms are compared against one another based on how many of the identified spam pages are contained within each bucket. Due to our interest only in the most reputable PLDs, we offer a different approach. We first assign reputation scores to all nodes, sort the graph by the score, and then scrutinize the top $R = 1{,}000$ domains in each ranking for spam. For each candidate in the list, we first attempt the PLD itself (e.g., http://amazon.com) and, if unsuccessful, consult the set of hostnames seen by IRLbot from that PLD to determine the shortest live website, which is then used as the representative for the domain. We follow all redirects and end up excluding 56 PLDs for which none of the websites are alive.

The second obstacle is that there is no consensus on the definition of web spam. Some researchers in the field classify all pornographic pages as spam [16], while others define a site as spam if it employs techniques intended to trick search engines into ranking some pages higher than they deserve [39]. While we generally aim to identify the latter types of PLDs, it is often difficult to detect cloaking, hidden links, and high-density farms/alliances pointing to some target page. We therefore rely on a subjective determination of what constitutes spam using two factors: attempts to perform malicious activities upon visit (e.g., install malware or viruses) or overwhelming presence of links whose main purpose is to create revenue from click-throughs. During our analysis, we commonly encounter link spam on *parked domains*, which may be expired PLDs for sale or those for which no website has yet been created. Domain-parking services fill these default pages with category-based links in hopes of turning a profit from click-throughs. Since these pages manipulate link-based ranking algorithms and serve no purpose other than providing sponsored redirects, we classify the corresponding PLDs as spam. While domain-parking companies often host pages for PLDs we consider spam, some of them also engage in legitimate activity such as domain registration, auctions, and web hosting (e.g., godaddy.com). In the end, we flag parking sites only if we believe their main purpose is to proliferate spam-like activity.

In addition to classifying each PLD as spam or not, we also record the *Google Toolbar Rank* (GTR) for each opened page, which we use to estimate the relative value of each PLD and achieve an additional level of granularity in the comparison.[2] Recall that Google offers a toolbar [34] that can be installed as a plugin to a user's web browser to provide, among other things, a rank between 0 and 10 for each visited page that reflects Google's opinion of the browsed document. The

[2]The GTRs used in this article were collected in 2008–2009. We have not repeated this activity to remain as close to the date of the original crawls as possible.

Table 1. Properties of the Datasets

Dataset	Crawled			Web graph		Host graph		PLD graph G	
	pages	hosts	PLDs	nodes	edges	nodes	edges	nodes	edges
IRLbot	6.3B	118M	33M	41B	310B	641M	6.8B	89M	1.8B
ClueWeb09	1.0B	31M	12M	9.3B	57B	118M	1.1B	31M	415M

toolbar returns no GTR for some pages, which occurs for resources that Google has not crawled, pages/domains that no longer exist or generate HTTP errors, and content purposely removed from the index. With the exception of [73], which studied a handful of GTRs for select Fortune 500 companies in 2003, we are not aware of any literature that has involved GTRs as a validation tool for unsupervised ranking methods.

3.3 Datasets

The first dataset is obtained from an IRLbot web crawl that took place in June–August 2007. During these 41 days, IRLbot pulled 8.2B pages from the queue, issued 7.6B requests, and successfully downloaded 6.3B HTML pages using a single server. The crawl started from a single seed (i.e., tamu.edu), kept both static and dynamic URLs in the frontier, and went to depth 31. The properties of this dataset are given in the first row of Table 1. The second crawl is ClueWeb09 [24], which contains 1B pages collected in January–February 2009. This crawl used Apache Nutch [59], parallelized using the Google-IBM-NSF cluster, and seeded off 33M trusted pages (e.g., DMOZ, query logs from AOL/Yahoo/Sogou) [26]. The original web graph available online [24] contained only 7.9B edges; however, when we parsed the 28TB of source HTML in ClueWeb09, we obtain a new graph that contained 57B edges. For reasons unknown, their crawler discarded all dynamic links (i.e., those with a ?) and downloaded only static pages. In the second row of Table 1, we document the properties of our version of ClueWeb09, which we use in the rest of the article due to its larger size than the original. Both graphs and our ranking results are available from [35].

These two datasets represent some of the largest academic crawls and the only mainstream efforts to use non-BFS exploration. They also exhibit a high degree of diversity, including separation by 18 months, usage of diametrically opposite strategies for selecting the seed set, and involvement of completely different frontier prioritization schemes. IRLbot used domain budgeting, where the nodes were ranked by PLD in-degree, and ClueWeb09 used OPIC at the page level. It is thus interesting to see if one ranking method can win against the alternatives on both datasets.

3.4 Top-Ranked PLDs

In general, one expects that a good ranking algorithm would place only unquestionably reputable domains near the top of the list. For IRLbot, Table 2 shows the top 15 PLDs of each candidate method and their GTRs. The first part of the table shows that IN's top results are all well-known domains and that 12 of them exhibit a GTR of at least 9. The next two lists, belonging to PageRank and OPIC, show a great deal of similarity to each other—both rank sedoparking.com in position 7 and information.com in the top 4. After much scrutiny, we label both of them as spam—the former heavily promotes parked domains filled with third-party advertising and is absent from Google's index; the latter is a revenue-marketing search engine hosting predominantly link farms. Further examination of the two PageRank-style algorithms reveals other suspicious PLDs (e.g., downloadrings.com, linkz.com) with low or missing GTRs, raising serious doubts about the ability of these two methods to prevent undesirable PLDs from achieving high ranks in large crawls. The

Table 2. IRLbot: Top 15 Ranked PLDs ("S" is Spam, "Q" is Questionable)

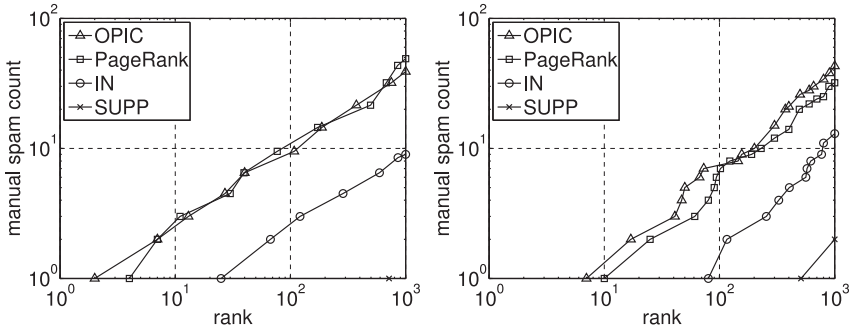
IN		PageRank		OPIC		SUPP	
microsoft.com	9	microsoft.com	9	microsoft.com	9	google.com	10
google.com	10	adobe.com	10	information.com (S)	5	microsoft.com	9
yahoo.com	9	google.com	10	google.com	10	yahoo.com	9
adobe.com	10	information.com (S)	5	adobe.com	10	adobe.com	10
blogspot.com	9	macromedia.com	10	macromedia.com	10	macromedia.com	10
wikipedia.org	9	yahoo.com	9	yahoo.com	9	wikipedia.org	9
w3.org	10	sedoparking.com (S)	–	sedoparking.com (S)	–	blogspot.com	9
geocities.com	9	googlesyndication.com	–	miibeian.gov.cn	9	msn.com	8
msn.com	8	w3.org	10	googlesyndication.com	–	apple.com	9
amazon.com	9	miibeian.gov.cn	9	w3.org	10	geocities.com	9
aol.com	8	downloadrings.com (S)	1	ndparking.de (Q)	–	w3.org	10
myspace.com	9	chestertonholdings.com (Q)	–	statcounter.com	9	sourceforge.net	9
macromedia.com	10	juccoholdings.com (Q)	–	searchnut.com (S)	–	youtube.com	9
youtube.com	9	statcounter.com	9	revenuedirect.com (Q)	4	bbc.co.uk	9
tripod.com	7	linkz.com (Q)	3	myspace.com	9	netscape.com	8

Table 3. ClueWeb09: Top 15 Ranked PLDs ("S" is Spam)

IN		PageRank		OPIC		SUPP	
google.com	10	adobe.com	10	adobe.com	10	microsoft.com	9
yahoo.com	9	google.com	10	google.com	10	google.com	10
blogspot.com	9	macromedia.com	10	w3.org	10	yahoo.com	9
adobe.com	10	yahoo.com	9	yahoo.com	9	adobe.com	10
wikipedia.org	9	w3.org	10	statcounter.com	9	wikipedia.org	9
youtube.com	9	blogspot.com	9	miibeian.gov.cn	9	blogspot.com	9
w3.org	10	statcounter.com	9	information.com (S)	5	youtube.com	9
amazon.com	9	microsoft.com	9	macromedia.com	10	apple.com	9
myspace.com	9	miibeian.gov.cn	9	blogspot.com	9	msn.com	8
wordpress.org	9	information.com (S)	5	myspace.com	9	geocities.com	9
microsoft.com	9	myspace.com	9	mapquest.com	9	macromedia.com	10
geocities.com	9	youtube.com	9	youtube.com	9	wordpress.com	9
miibeian.gov.cn	9	wikipedia.org	9	wordpress.org	9	aol.com	8
wordpress.com	9	amazon.com	9	amazon.com	9	w3.org	10
statcounter.com	9	mapquest.com	9	microsoft.com	9	amazon.com	9

final two columns show the top domains from SUPP. There is a substantial overlap with the IN list, but SUPP is slightly better—13 PLDs with a GTR at least 9 and all 15 with at least 8.

As shown in Table 3, ClueWeb09 results generally look better since the crawl was seeded with a large amount of good URLs, executed on a smaller scale, and reached a significantly lower depth. It is interesting, however, that information.com still manages to rank high in PageRank and OPIC, which is peculiar given the different prioritization methods used in these crawls and 1.5 years in between. The SUPP ranking of ClueWeb09 is pretty similar to that in IRLbot, with the top 4 nodes being exactly the same, albeit in different order. There are three new domains in positions 12, 13, 15, i.e., wordpress.org, aol.com, and amazon.com, but the remaining nodes are the same.



(a) IRLbot (b) ClueWeb09

Fig. 1. Manual spam count.

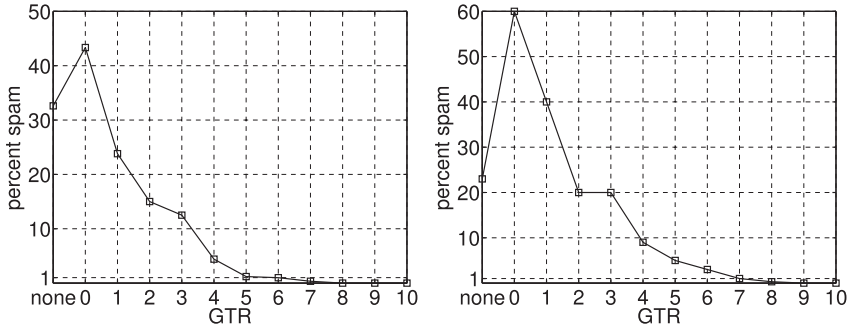
3.5 Spam Avoidance

In Table 2, it is relatively easy to compare the methods; however, the situation is less obvious in Table 3, where all four techniques produce reasonable ranking if we ignore outlier information.com. It is thus necessary to create analysis techniques that can reveal ranking quality in a more clear fashion. We offer such an approach next. Define u_r to be the number of spam PLDs in positions $[1, r]$ within a given ranking list. Note that u_r is a CDF-like function that starts from 1 at the first spam domain and monotonically increases thereafter. The plot of u_r for IRLbot is shown in Figure 1(a). What stands out first is that PageRank and OPIC accumulate spam at approximately the same rate as r increases, despite their huge differences in computational complexity. They are also significantly inferior to the other two methods. While IN allows only nine spam PLDs into the top 1K list of IRLbot (i.e., 4.5× fewer than PageRank/OPIC), the real winner in this comparison, however, is SUPP, whose entire top 1K list contains only one spam domain (linksynergy.com in position 718). Results for ClueWeb09 are shown in Figure 1(b). They are similar to IRLbot's, except IN/PageRank/OPIC push their first spam PLD further down the list. In the end, however, they end up with similar or slightly worse totals as in IRLbot, which is caused by the steeper growth rate of spam. SUPP produces two spam domains (pathfinder.com in position 510 and linksynergy.com in position 868) but still comes out clearly on top.

To fully understand the implications of Figure 1, recall that domains ranked near the top may be allocated a gigantic budget during crawling. We now ballpark these numbers from publicly available Google data. Using site queries from [48], we found that the top 1K domains in the SUPP ranking were collectively responsible for 18% of Google's 2008 index (i.e., 5.5B pages out of 30B). Considering that these 1K domains represent only 0.001% of the PLDs known to IRLbot (and probably even less among those known to Google), it becomes clear that top-ranked domains receive a heavy bias, i.e., on average 30,000× more budget than domains outside the top list. Spam that manages to infiltrate the top 10 list, as it does for OPIC/PageRank in IRLbot, can cause significant resource wastage and noticeably detrimental effects on the collected web graph.

3.6 GTR and Spam

We finish this section by examining how well GTR values predict the occurrence of spam in the domains examined manually. Using the IRLbot dataset, Figure 2(a) shows the percentage of PLDs with a given GTR that are spam according to our manual analysis. The peak in the curve occurs at 43% for GTR-0 PLDs, which is a significant fraction. It is also likely that this number should be higher given the rather conservative definition of spam used in our decisions. What can be clearly



(a) IRLbot (b) ClueWeb09

Fig. 2. Spam probability.

stated based on manual inspection of GTR-0 PLDs is that none of them are well-known, reputable domains. In fact, many of them are suspicious (e.g., related to gambling or pharmaceutical sales), while others are simply obscure. This suggests that Google appropriately applies GTR = 0 to sites that have little download value. It is also interesting to examine PLDs not ranked by Google, which are responsible for the second-largest percentage (i.e., 32%) in Figure 2(a). Our inspection shows that the majority of these PLDs are again either spam or highly questionable. Following the rest of the curve, notice that it monotonically decays and becomes zero for GTRs larger than 7. In fact, almost no spam is contained in PLDs with a GTR of 5 or higher (i.e., less than 0.6%). Spam likelihood for ClueWeb09, shown in Figure 2(b), exhibits a similar overall trend. Thus, our general conclusion from this activity is that toolbar ranks accurately reflect domain value, at least in some average sense.

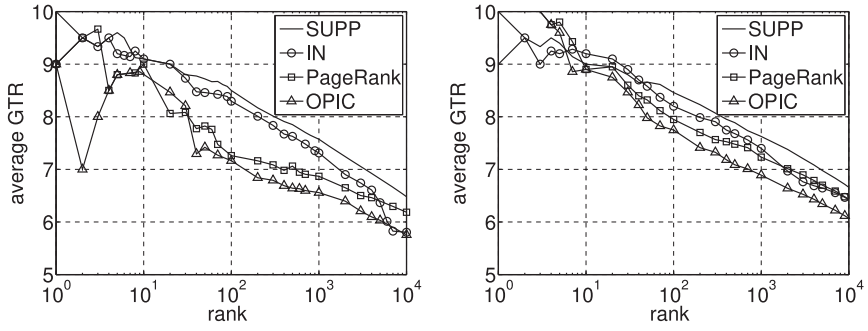
Since bulk GTR information is publicly available in certain portals on the web, we next leverage these findings to create an automated method for comparing ranking algorithms. Note that GTRs cannot be used during crawls because this information cannot be possibly acquired in real time without getting blocked by the GTR service. Additionally, GTR data is neither guaranteed to exist in the future nor fine-granular enough to establish a usable ranking of domains (e.g., millions of PLDs have the same integer GTR score). Since our comparison needs only a few thousand offline lookups for the top R domains, this strategy works well for our purpose.

4 AUTOMATED ANALYSIS

4.1 Average GTR

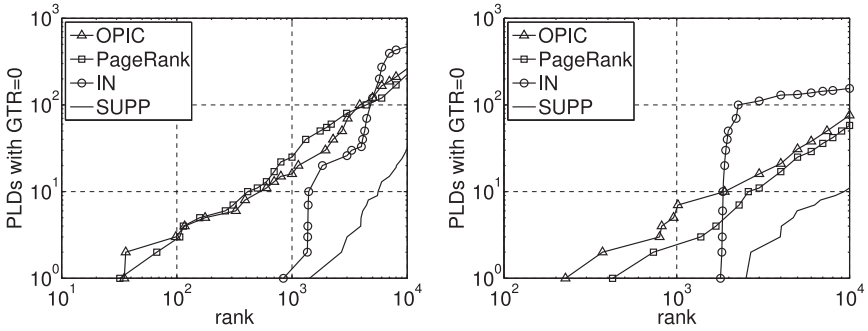
We start by determining the ability of the studied algorithms to place the most valuable PLDs at the top of the list. Figure 3(a) plots a moving average of the numeric GTR in IRLbot. The graph shows that SUPP maintains the highest average GTR, while IN follows closely until approximately position 4K. It then exhibits a sharp drop caused by a large number of domains related to worldnews.com, which all have zero GTR. These pages contain links to valid news articles and appear legitimate on the surface, except for their odd linking structure. Instead of using a single PLD with several subdirectories or multiple hosts to organize the site, their content is divided among hundreds of PLDs, all of which link to one another in a virtually complete graph. This tight link structure may appear as spam to Google and may be the reason they are branded with GTR 0.

The authors in [31] observe a similar case that arises from *link exchanges*—each user j buries links to $N-1$ other member sites, which in return host URLs back to j. Link exchanges inflate the in-degree of each participant to $N-1$ using just N nodes in the graph and sometimes involve



(a) IRLbot (b) ClueWeb09

Fig. 3. Average GTR.



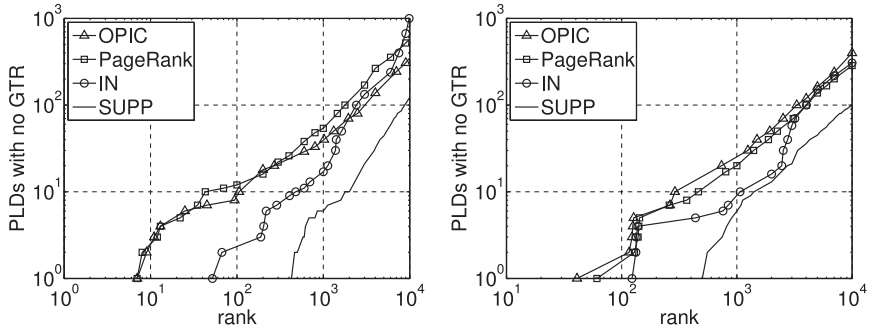
(a) IRLbot (b) ClueWeb09

Fig. 4. GTR-0 PLDs.

thousands of unique PLDs. They harvest high in-degree at low cost and successfully manipulate certain ranking schemes, such as IN. The SUPP algorithm aims to overcome this exact problem. Since all nodes in a link exchange have distance 1 from each other, their self-induced SUPP count is zero. This allows SUPP to keep all worldnews.com domains out of its top 10K, which is aided by the fact that constructing a large enough link farm to be competitive with reputable sites at distance $D = 2$ requires many more resources (i.e., not thousands, but millions of unique PLDs). The other two methods in Figure 3(a), i.e., PageRank and OPIC, stay noticeably lower and again track each other. The only major difference is that OPIC takes hits from information.com and a few other questionable PLDs near the top of its ranking. Interestingly, the same qualitative conclusions hold for ClueWeb09 in Figure 3(b), where IN suffers from the worldnews.com clique around position 2K.

4.2 Weak GTR

While Figure 3 is a good starting point, it does not reveal the details hidden behind each curve. We therefore now focus on function u_r, which counts the various undesirable domains in the top list. Figure 4(a) plots the cumulative distribution of GTR-0 PLDs in each list of IRLbot. SUPP stays flawless until home.net (a link-spam site) shows up in position 1,422. It finishes with 32 GTR-0 PLDs in the top 10K. IN initially does an excellent job of suppressing unwanted domains, with only home-equity-loans-1.org in the top 1K at position 843; however, around $r = 1.4$K, zero GTR domains start rapid accumulation. This is further exacerbated at $r = 4$K, where worldnews.com



(a) IRLbot (b) ClueWeb09

Fig. 5. Unranked PLDs.

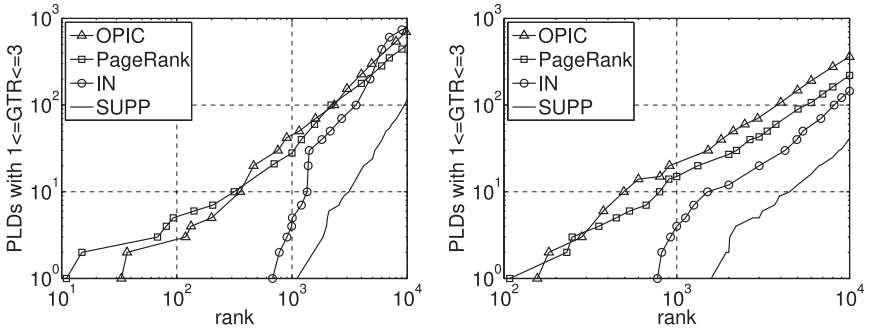
-related PLDs spike up the totals even higher. After position 5K, IN performs the worst in this comparison, allowing 448 GTR-0 domains in the top 10K. OPIC and PageRank are again pretty similar, with the former slightly outperforming the latter. Both have zero-ranked PLDs high in their list—everytihng.com in position 32 for PageRank and home-equity-loans-1.org in position 35 for OPIC. For ClueWeb09 in Figure 4(b), SUPP admits only 11 bad domains in the top 10K list, where the first one (i.e., msplinks.com) appears in position 2,517. IN works well until $r = 2$K, where the same worldnews.com cluster sharply increases its count u_r to 100. OPIC/PageRank admits more GTR-0 domains in the top 1K than the other two methods, finishing with 8× more spam than SUPP.

Figure 5(a) examines the presence of PLDs without a GTR in IRLbot. SUPP continues to maintain a noticeable distance from the others, with only five unranked domains in the top 1K and 115 in the top 10K. Its first no-GTR domain bfast.com shows up in position 469. Early on, IN splits the difference in performance between SUPP and PageRank-style methods, with just 17 unranked PLDs in the top 1K. Its first unranked domain is googlesyndication.com in position 52, which is used by Google for hosting its AdSense. Even though many pages link to it, it stores no crawlable content, which explains the absence of a GTR. IN eventually ends up with 1,059 unranked PLDs in its top 10K, surpassing all other techniques near the end. PageRank and OPIC curves are again similar, both pushing four unranked PLDs in the top 15 (see Table 2), and respectively admitting 39 of 54 such domains in their top 1K and 329 of 556 in their top 10K. Figure 5(b) illustrates similar results from ClueWeb09.

Besides the two categories of GTR just considered, Figure 2 suggests that GTR scores below 4 are also indicative of domains that should not be massively crawled. We thus count PLDs with a GTR between 1 and 3 and show the result in Figure 6. SUPP again stands out with a 4 to 5× better performance than the closest competitor, keeping low-GTR domains completely out of its top 1K. IN takes second place with only four such PLDs. However, after that, it quickly loses its advantage and ends up with performance close to that of PageRank/OPIC.

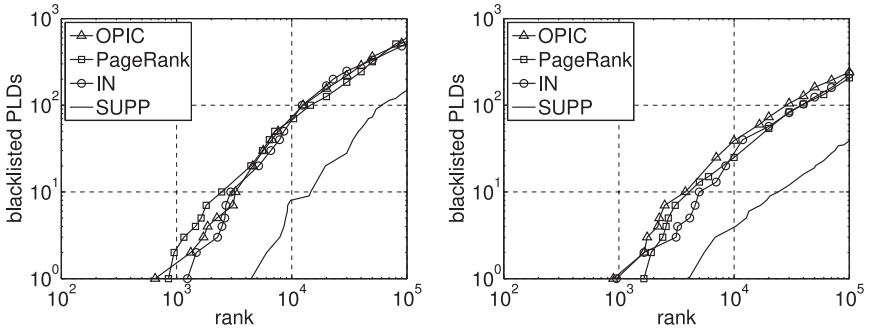
4.3 Blacklisted Domains

In addition to GTRs, there are publicly available blacklists that contain domains related to email spam (e.g., PLDs that originate email spam, host phishing content, distribute scams or viruses). These may or may not contain the kinds of link spam we commonly aim to avoid, but the fact that they are involved in proliferation of unsolicited bulk email suggests that they may also employ other unethical techniques, e.g., web spamming or rank manipulation aimed to increase the visibility of their site. We use a SpamAssassin blacklist [69] with 496,698 unique domains. To give an



(a) IRLbot (b) ClueWeb09

Fig. 6. Low-GTR PLDs.



(a) IRLbot (b) ClueWeb09

Fig. 7. Blacklisted PLDs.

idea of its coverage, this dataset contains 60,701 PLDs seen by IRLbot. We find their occurrence in each ranking list and plot the corresponding count in Figure 7(a). SUPP again does the best job of avoiding blacklisted domains, maintaining a clear separation from the other lists up through $R = 100$K. The first blacklisted PLD in SUPP ranking is topmeds10.com in position 4,459 and only seven appear in the top 10K. IN show the next-best performance with its first undesirable PLD yourmedpharm.info in position 1,246. PageRank/OPIC predictably finishes last, both with blacklisted domains in the top 1K. In Figure 7(b), ClueWeb09 enjoys similar results, where SUPP is 10× better in the top 10K and 5× in the top 100K, while the other three methods are essentially identical to each other.

4.4 Reputable Domains Ranked Low

We have shown that SUPP decisively beats all studied alternatives. To understand its false negatives, we analyze whether any good domains (i.e., with GTRs 9–10) have inadvertently ended up at the bottom of its ranking list. In each dataset, we discover approximately 450 such PLDs that are ranked beyond the top 10K. After manual examination, *all* of these cases can be classified into the following four categories: (1) synonyms, which include (a) misspelled names of famous companies (e.g., amozon.com, verisgn.com); (b) unknown PLDs that redirect to Google, Adobe, Yahoo, or some equally famous PLD (e.g., hospitalquality.org, defytherules.com, vknn.org, floralartbyamy.com); (c) country versions of well-known sites (e.g., reuters.cz); (2) mirrors that do not redirect through HTTP 301/302 but visually look identical to their main site (e.g., columbiauniversity.net,

compap.com, cnn.co.il); (3) sites hosted under .gov, .edu, and .mil, whose GTR is presumably inflated by Google based on their TLD (e.g., ucaid.edu, volunteer.gov, pentagon.mil); and (4) GTR anomalies that have since been corrected by Google and whose current GTR is much lower, usually 2 to 5 or no GTR at all (e.g., laraweb.com.ar, absearecruitment.com, baileychevy.com). The issue with TLDs that enforce strict registration rules (e.g., colleges for .edu, government offices for .gov) can be solved by statically assigning high ranks to them, even if they do not have a sufficient number of incoming links in the crawled portion of the web. In fact, it is possible that Google already does that, which explains its high GTR. The remaining GTR 9 to 10 outliers are not worth crawling, either because they contain duplicate information or because they are not reputable in the first place; i.e., no useful content will be lost by avoiding them in the future.

5 SUPERVISED RANKING WITH AUTOGENERATED LISTS

Our previous comparison considers only unsupervised techniques that calculate ranks strictly based on the link structure of the PLD graph and without any knowledge about the nodes being ranked. We now examine whether their performance can be improved using supervised methods; however, to keep the cost of manual involvement similar, we only consider algorithms that automatically assign labels to nodes in the graph and do not require extensive maintenance.

5.1 TrustRank

Recall that the random walker in the PageRank model (Equation (2)) has an equal probability of teleporting to any page. Thus, even pages with little or no value still receive a small PageRank score. This allows spammers to aggregate PageRank probabilities of multiple pages to successfully influence the rank of arbitrary targets [31, 38, 39, 64]. TrustRank [40] addresses this problem by incorporating a *whitelist* W of trusted pages, which are used as uniformly random destinations of jumps. Defining the trust of page j as

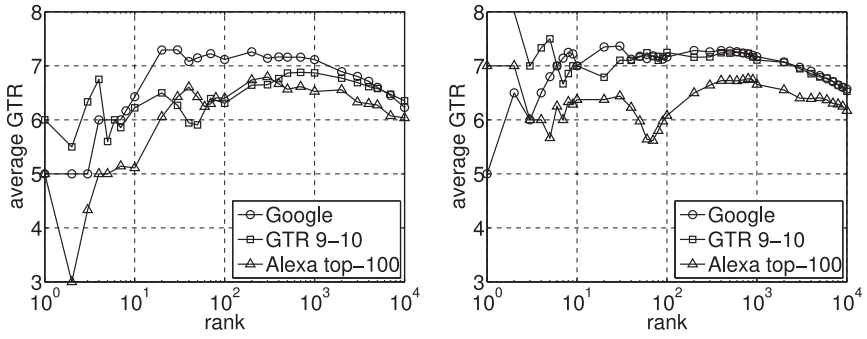
$$t_j = \begin{cases} 1/|W| & j \in W \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

the PageRank model in Equation (2) gets a revision in the additive constant to focus all teleportation to the nodes in W, i.e., $\epsilon_j = (1 - \alpha)t_j$.

The main caveat with TrustRank is that there is no universally accepted methodology for generating trusted PLDs. The original authors suggest [40] that the whitelist should consist of nodes with a high out-degree so that the algorithm can converge quickly, in addition to containing as many reputable sites as possible. Balancing this with an objective to produce W with minimal human interaction, our first attempt followed this recommendation and ordered the PLDs by their out-degree, among which we selected the top 1K domains with a GTR of at least 7. The reasoning was that PLDs highly regarded by Google should be trusted; however, the corresponding TrustRank outcome on IRLbot was quite poor and contained many low-value PLDs near the top, which was worse than even OPIC. Since the problem of choosing a good W appears nontrivial, we next perform a separate investigation into it.

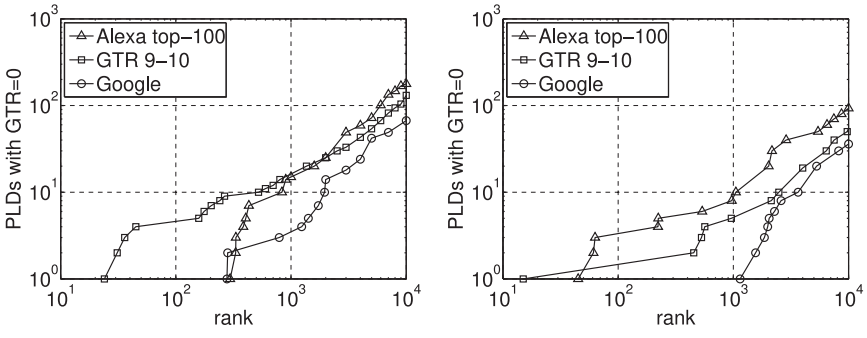
5.2 Whitelist Selection

It should be first noted that direct comparison of TrustRank with different W may produce misleading results. The problem arises because the whitelisted domains trivially end up ranked at the top of the list, which provides an advantage to cases with larger W. To remove this bias, the experiments below exclude the whitelisted PLDs from the rankings and renumber the remaining domains up to position R. We then compare each list by the same criteria as before.



(a) IRLbot (b) ClueWeb09

Fig. 8. Impact of whitelist selection on average GTR.

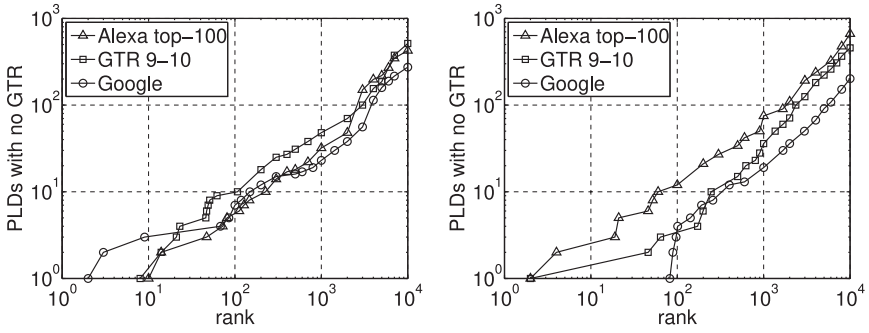


(a) IRLbot (b) ClueWeb09

Fig. 9. Impact of whitelist selection on GTR-0 PLDs.

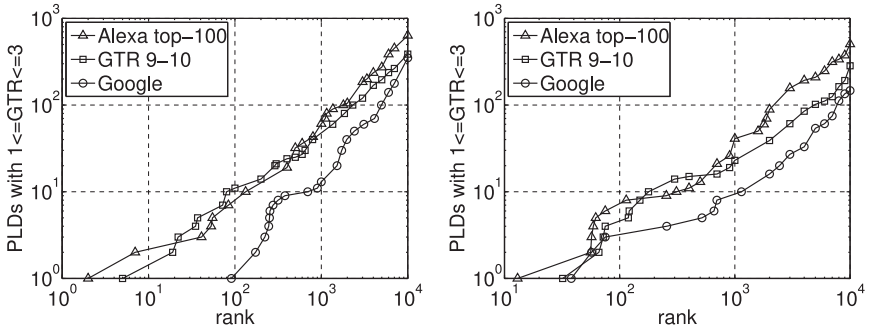
We abandon the idea of using out-degree as an indication of quality and choose W using three strategies—a single highly reputable domain (i.e., google.com) as the only trusted node, Alexa Top 100 PLDs [4], and all domains with a GTR of 9 to 10 from our datasets. While all three methods can be easily automated, only the first one may be considered somewhat equivalent to unsupervised techniques studied earlier. The other two lists require access to external databases, which raises the possibility that the crawler may eventually be blocked due to high frequency of access and the service may be removed altogether. Nevertheless, these whitelists should provide a good indication of whether a more expensive-to-obtain and longer W should be pursued for TrustRank in future crawler endeavors.

Figures 8 through 12 examine the average GTR and amount of undesirable domains in the ranking of each candidate algorithm. Among the 10 plots presented here, Google outperforms the other two alternatives in nine cases. The only exception is Figure 12(b), where it is inferior to GTR 9 to 10. Another consistent result is that Alexa Top 100 is the worst whitelist in this comparison. One possible reason that longer lists work poorly is that TrustRank boosts direct out-neighbors of W, which themselves are *not* always reputable domains. This is a consequence of the fact that whitelisted PLDs are not required to link to other highly ranked nodes; in fact, larger W allows for more potentially shady domains to climb up in ranking. As a result, selecting a large whitelist (thousands of nodes) that justifies the extra complexity/cost of its acquisition will likely remain an open problem in this area.



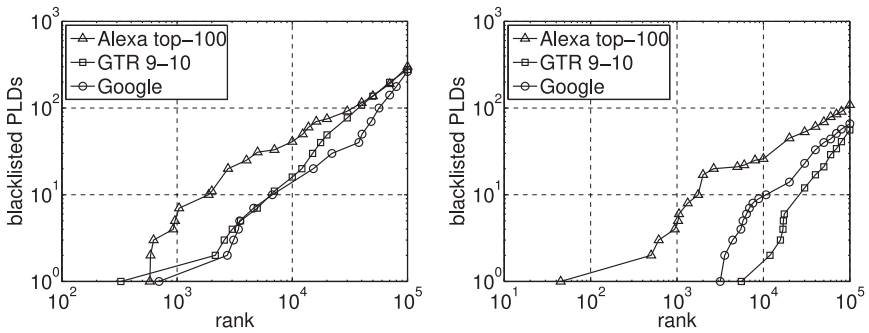
(a) IRLbot (b) ClueWeb09

Fig. 10. Impact of whitelist selection on unranked PLDs.



(a) IRLbot (b) ClueWeb09

Fig. 11. Impact of whitelist selection on low-GTR PLDs.

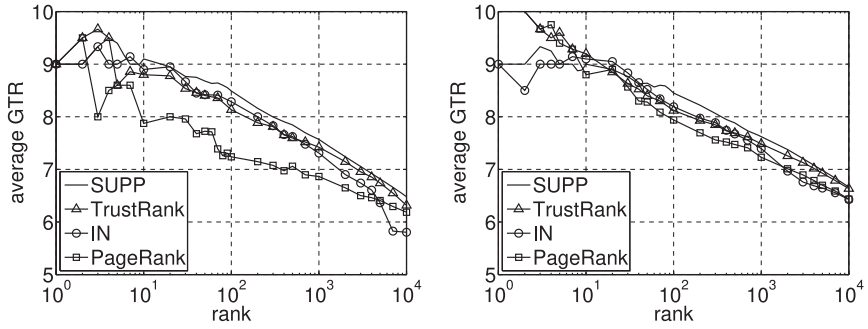


(a) IRLbot (b) ClueWeb09

Fig. 12. Impact of whitelist selection on blacklisted PLDs.

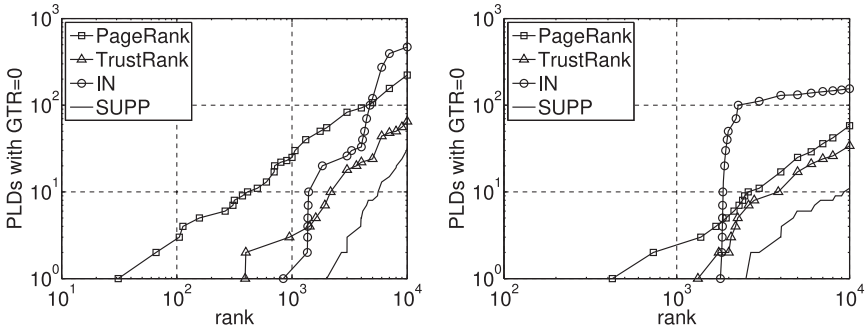
5.3 Relative Performance

Even though TrustRank-Google beats the other whitelists, it is unclear how it stacks up against unsupervised methods used in earlier sections. To shed light on this issue, we compare its performance against PageRank, IN, and SUPP, omitting OPIC to avoid clutter. It should be noted that to enable fair comparison, google.com is removed from all results, which makes the average GTR plots of unsupervised algorithms slightly different from those seen earlier. Figures 13 through 17



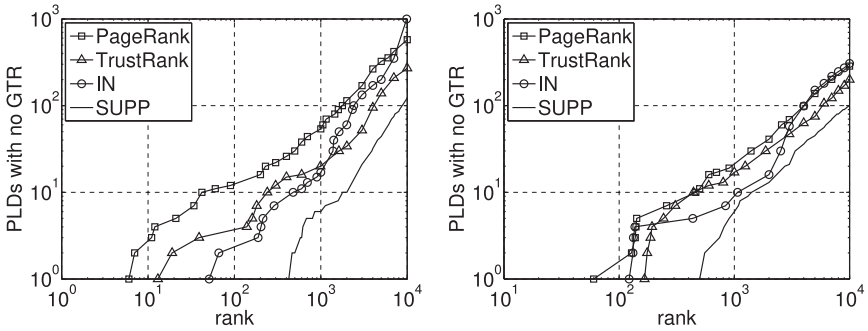
(a) IRLbot (b) ClueWeb09

Fig. 13. Comparison with TrustRank on average GTR.



(a) IRLbot (b) ClueWeb09

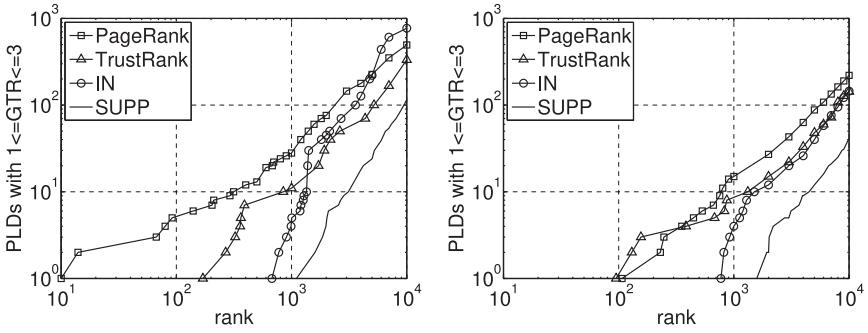
Fig. 14. Comparison with TrustRank on GTR-0 PLDs.



(a) IRLbot (b) ClueWeb09

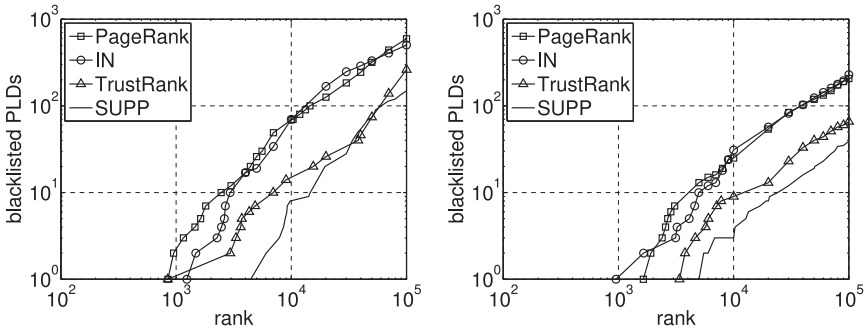
Fig. 15. Comparison with TrustRank on unranked PLDs.

show that TrustRank achieves a significant improvement over PageRank, outperforming it in the majority of cases. Moving through the plots, TrustRank and IN alternate in taking second place, neither decisively winning over the other. Interestingly enough, SUPP still demonstrates a clear advantage across all cases and datasets. It keeps a substantial distance to the nearest competitor, except in Figure 17(a), where it touches the TrustRank curve after position $r = 30$K. Considering that SUPP works without knowing the meaning of individual nodes, it is possible that it can



(a) IRLbot (b) ClueWeb09

Fig. 16. Comparison with TrustRank on low-GTR PLDs.



(a) IRLbot (b) ClueWeb09

Fig. 17. Comparison with TrustRank on blacklisted PLDs.

produce excellent ranking even for other types of graphs (e.g., host level) and be beneficial in scenarios beyond frontier prioritization (e.g., analysis of social networks). Given its generality and potentially wider applicability in the future, our next objective is to investigate how to use SUPP in practice.

6 ESTIMATING TOP SUPPORTERS

From the previous sections, it is easy to see that SUPP produces the best-ranked PLD lists. However, calculating exact supporter counts does not scale well because of the enormous amount of CPU processing required to perform a limited-scope BFS flood from each node in the graph. However, the good news is that our problem does not require SUPP counts for poorly supported nodes; instead, a high-performance crawler is interested in a *fast, accurate, and scalable technique for estimating supporters at the top of its ranking list*. We offer such an approach below.

6.1 Depth of Supporters

We start by addressing the issue of whether depth $D = 2$ is the correct choice for Internet-wide PLD graphs. Exploring $D = 3$, we find it to be a poor indicator of site reputation. As an illustration of this, Table 4 lists the number of supporters at depth 1 to 10 for two PLDs with drastically different reputations. Google is ranked by SUPP as #1 in IRLbot and #2 in ClueWeb09, while hotsiteskey.info is in positions #1,698,361 and #2,951,371, respectively. Following the rows of the table, google.com

Table 4. Supporter/Depth Analysis

	IRLbot		ClueWeb09	
Depth D	google.com	hotsiteskey.info	google.com	hotsiteskey.info
1	2.2M	2.32K	1.0M	1
2	15.5M	1.7M	6.1M	650K
3	6.2M	15.6M	1.7M	5.8M
4	731K	6.6M	214K	2.3M
5	62.8K	752K	23.1K	285K
6	7.40K	69.8K	3.12K	29.4K
7	1.30K	7.97K	334	3.82K
8	456	1.26K	60	530
9	244	519	29	72
10	189	275	23	25

starts off with orders of magnitude more supporters at levels 1 and 2; however, it falls behind starting at depth 3. There are two reasons behind this effect, which we discussed next.

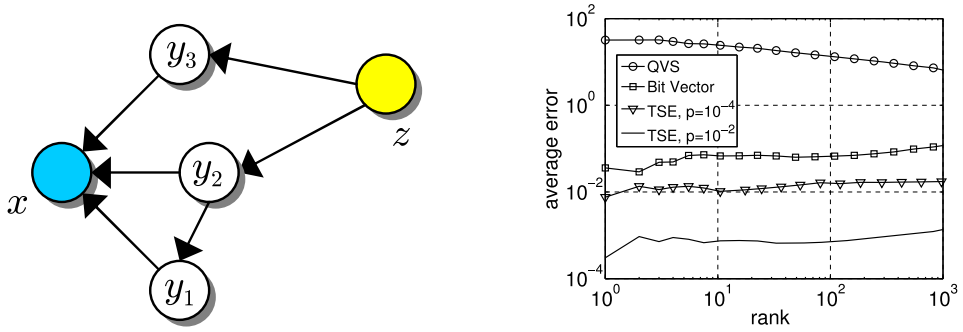
The first one is related to how quickly the graph expands during BFS and how many (or few) nodes are reachable at larger depths. The average in-degree $E[X_i]$ is quite small—20.2 for IRLbot and 13.5 for ClueWeb09; however, the average level 2 supporter count is a massive number—21K for IRLbot and 110K for ClueWeb09. It may be surprising that both are much bigger than the square of average in-degree. This drastic difference comes from the fact that SUPP counts are proportional to $E[X_i^2]$, which can be much larger than $E[X_i]^2$. Therefore, PLD graphs exhibit a rapid explosion of supporter counts for popular PLDs and lack nodes for them to reach at larger depth. The second reason is that hotsiteskey.info manages to attract links from adobe.com in IRLbot and blogspot.com in ClueWeb09, whose SUPP counts are very similar to Google's. This explains why the total score of hotsiteskey.info at distance $D+1$ closely follows that of Google at depth D. Knowing that Google supporters peak at depth 2, it is no wonder that it loses at every level afterward. All of this suggests that SUPP with $D \geq 3$ may be potentially appropriate for slowly branching graphs (i.e., those with much larger average distances), but clearly not the Internet PLD graph.

6.2 Analysis

To understand the real-time infeasibility of SUPP, we start by analyzing its basic complexity under the assumption that the graph fits in RAM. We later extend this to external memory scenarios. As shown in Figure 18(a), SUPP for each node x must count the number of unique nodes z whose shortest path to x is exactly two hops, which is usually accomplished by a BFS along the in-links. To calculate the overhead of SUPP, we need to define a new metric we call *Quick-Visit Supporters* (QVS) that counts the number of link traversals during BFS:

$$Q_j = \sum_{d(i,j)=1} X_i, \tag{4}$$

where X_i is the in-degree of node i. Note that QVS can also be viewed as an extension of IN to two iterations in Equation (1). QVS measures all nodes reachable in two steps, including multiple visits to the same node and those that might be at shortest distance 1. Figure 18(a) shows an illustration. Observe that $SUPP(x) = 1$, but $Q_x = 3$ since QVS counts y_2 once and z twice. Even though there is a path $y_2 \rightarrow y_1 \rightarrow x$, node y_2 is not a level 2 supporter since its shortest distance to x is 1. Elimination of duplicates and counting *unique* nodes z is what makes SUPP slow. A depth-2 BFS from all nodes



(a) SUPP (b) Average error (IRLbot)

Fig. 18. BFS-based SUPP and estimation error.

requires $n(E[Q_x] + E[SUPP(x)])$ nonsequential RAM hits to mark all visited nodes and then to clear the set bits. In IRLbot, $E[Q(x)] = 34.4$K and $E[SUPP(x)] = 21$K, which means that SUPP must issue almost 5 trillion random memory lookups. Using 60ns RAM latency, this amounts to 82 hours of number crunching. CPU caching makes the result slightly faster, but computation still takes almost 3 days to finish. More strikingly, ClueWeb09 exhibits $E[Q_x] = 138$K and $E[SUPP(x)] = 110$K. Even though its graph is 4.4× smaller than IRLbot's and less dense in terms of edges, its second moment of in-degree $E[X_i^2]$ is much larger. As a result, SUPP ranking of ClueWeb09 requires a whopping 7.6 trillion random hits.

One approach is to approximate SUPP using QVS, which runs a lot faster and obtains the estimates in just $nE[X_i]$ lookups (i.e., 1.8B for IRLbot and 414M for ClueWeb09). Another method is *Bit Vector* (BV) [10], in which nodes iteratively receive bit strings from their in-degree neighbors and apply a bitwise OR operation to them. Since OR is not additive, this process avoids the main pitfall of QVS, i.e., multiple counts of the same node z. BV requires $2rnE[X_i]$ lookups in RAM, the same number of OR operations, and generation of rbn random numbers, where $b = 64$ is the number of bits in each vector, $r = \log_2(S_{max})$ is the number of performed rounds [10], and S_{max} is the maximum SUPP count. This leads to 25 rounds to converge in the IRLbot case with $S_{max} = 15.5$M and 23 rounds in the ClueWeb09 case with $S_{max} = 6.1$M. However, since we are only interested in the top list, BV can be adapted to terminate earlier by performing only the last few rounds (i.e., two for the top 1K and three for the top 10K).

6.3 Top Supporters Estimation (TSE)

We next develop a new method we call *Top Supporters Estimation (TSE)*, which allows efficient approximation of SUPP counts for nodes with a large base of supporters. First intuition suggests to visit all in-neighbors $\{y_i\}$ of x and then randomly subsample *their* in-neighbors $\{z_j\}$ with probability p, which can be done efficiently by skipping through neighbor lists using a geometric random variable. This approach, which we call *naive*, performs almost as fast as QVS; however, there unfortunately is no good way to reconstruct the number of *unique* supporters from the collected samples. It is well known that given a multiset Γ (i.e., a set of items with repetition), a p-percent blind subsample of Γ cannot be used to accurately determine the number of unique items in Γ. This problem has been studied extensively in databases [18] and network flow sampling [28], without much promise for a solution. To illustrate why this happens, consider an example. Suppose x generates $\Gamma = \{1, 2, 2, 5, 5, 5, 7, 10, 10, 11\}$. The exact algorithm (i.e., SUPP) removes duplicates from this list and obtains a smaller set $S = \{1, 2, 5, 6, 10, 11\}$ of unique supporters. On the other hand, the naive estimator performs a p-percent subsample of Γ. For $p = 1/2$, this may produce a new list

$\Gamma' = \{1, 2, 5, 7, 10\}$, which cannot be used to recover the size of S with any sort of reasonable accuracy. This comes from the fact that each node in Γ is repeated with an a priori unknown frequency. Of course, if one is allowed to examine each item in Γ before deciding whether to sample it or not, accurate estimators exist [74]; however, in such cases BFS has a similar CPU overhead and there is no need for an estimator.[3]

Instead, we use a novel two-pass technique that leverages both in/out-graphs. Using the notation of Figure 18(a), the first phase scans the out-graph and randomly retains in RAM a p-fraction of all nodes z with their entire out-neighbor adjacency lists $\{w_j\}$. This produces an unbiased random sample of all supporters z that x will later count. Because subsampling applies to a list of unique items $\{1, \ldots, n\}$, there is no bias toward nodes with a larger degree as in the naive method. Then, the second phase reads sequentially the in-graph, examining each node x with its in-neighbors $\{y_i\}$. Assuming $x \neq z$ and $x \notin \{w_j\}$, any overlap between sets $\{w_j\}$ and $\{y_i\}$ indicates that z supports x at level 2. This detection is accomplished using hash tables with efficient multi-CPU parallelization. If z discovers that it supports x, it increments x's counter by 1. After all sampled PLDs z have been processed, the supporter total of x is scaled by $1/p$, the result is written to disk, and the next node x is read from the in-degree file. For the example above and $p = 1/2$, assume that the six original supporters of x are subsampled to $\{1, 5, 10\}$. After SUPP is executed on the modified graph, it finds $\Gamma = \{1, 5, 5, 5, 10, 10\}$, removes duplicates, and arrives at a total of $|S| = 3$ supporters. Scaling this by $1/p$, we get the correct count of six.

While the algorithm can work on one x at a time, in practice files are processed in chunks large enough to keep I/O efficient. Neglecting certain small terms, TSE requires RAM lookup overhead $pn(E[Q_x] + E[SUPP(x)])$, which is a p-fraction of that of full BFS. It should also be noted that since the crawler first builds the out-degree PLD graph as it parses pages, which is later inverted to obtain the in-degree PLD graph (used by all other methods), there is no additional cost to obtain the two graphs needed for TSE's operation.

6.4 Comparison

We now examine the various RAM-only algorithms in terms of accuracy and runtime. For these experiments, we use 64 bits for each node in BV and the adaptive approach recommended in [10]. We also retain level 1 BV vectors and remove any overlap with level 2 supporters so as to exclude immediate neighbors of each node x. For this comparison, the PLD graph is processed offline to re-write its 8-byte hashes using sequential 4-byte labels. Nonsequential, larger IDs result in higher RAM consumption, less CPU cache locality, and more overhead for each algorithm. Such cases are not considered here, but they are straightforward extensions of our results below.

Using SUPP's order of ranking for the top domains, Figure 18(b) shows a sliding-window average of relative error for each estimation technique as a function of the node's rank on the IRLbot dataset. Observe that QVS exhibits enormous error (i.e., over 1,000%) for the very top PLDs, which gradually reduces to about 200% near rank 1K. In second place is BV, whose error averages 6.5% across the whole plot, which is quite a bit better than the 15% to 17% found in [10]. However, TSE's error is even lower, ranging from about 1% for $p = 10^{-4}$ to 0.1% for $p = 10^{-2}$. In fact, no TSE errors exceed 0.6% in the top 1K for $p = 10^{-2}$ and 7.2% for $p = 10^{-4}$, while the worst BV error is 28% and that of QVS is 3,200%. In ClueWeb09, the results are similar.

Table 5 shows the theoretical number of random RAM hits and various CPU operations (e.g., additions, ORs, random numbers generated) in each method. It also displays the actual runtime of these algorithms on a quad-core AMD Opteron server (disk I/O is excluded). Each algorithm is

[3]Typical reasons for subsampling Γ are memory-related restrictions. In our problem, just performing a random memory access for each item in set Γ incurs prohibitive CPU overhead.

Table 5. Memory Hits, CPU Operations, and Runtime
(2.8GHz Quad-Core Opteron)

	IRLbot (7.9 GB)			ClueWeb09 (1.8 GB)		
Algorithm	Hits	Ops	Time	Hits	Ops	Time
SUPP	4.9T	1.9T	70 hr	7.6T	3.4T	108 hr
TSE ($p = 10^{-2}$)	49B	19B	11 min	76B	34B	17 min
Bit Vector ($r = 2$)	7.1B	11B	3.8 min	1.6B	3.9B	51 sec
TSE ($p = 10^{-3}$)	4.9B	1.9B	70 sec	7.6B	3.4B	1.8 min
QVS	1.8B	1.8B	55 sec	414M	414M	13 sec
TSE ($p = 10^{-4}$)	490M	190M	7.5 sec	760M	340M	11.6 sec

multithreaded and optimized to run with enough memory to hold the entire PLD graph and each algorithm's own data structures. On IRLbot, SUPP slugs along for 70 hours, occupying all four cores at 100% utilization. This is 35× slower than 50-iteration PageRank. The default BV method (which requires over 20 rounds) runs for 47 minutes; however, its scaled version that produces ranking only for the top 1K nodes (two rounds) terminates in 3.8 minutes, which is 3× quicker than the most accurate version of TSE considered here (i.e., $p = 10^{-2}$). The other two TSE configurations in the table run much faster, with $p = 10^{-4}$ beating even QVS. On ClueWeb09, SUPP spends over 4 days; however, BV terminates sooner here than in IRLbot because its complexity is proportional to $E[X_i]$, which is smaller in ClueWeb09.

Interestingly, TSE with $p = 10^{-4}$ is more accurate than BV not just for the top 1K, but also for the top 10M domains. Thus, if estimates for such a large top list are needed for some reason, TSE can be a more impressive alternative to BV since the latter now requires 13 iterations to obtain convergence for the top 10M, which increases its runtime sixfold compared to Table 5. Despite these findings, BV has merit in its ability to tackle $D \geq 3$ with little additional overhead, while TSE may require significantly lower p as D increases to control the explosion of BFS. As we encounter future graphs that require SUPP estimation at such depths, we will revisit this issue and study TSE's accuracy in those settings against BV's.

6.5 External Memory

We finish by discussing handling of large graphs that do not fit in RAM, where the main performance metric is the amount of disk I/O (i.e., the CPU cost is neglected). The most straightforward method, which we call SUPP-A, loads a sequential chunk of vectors $(x, y_1, y_2, \ldots)$ from the in-graph into some buffer B and then rescans the entire in-graph to check all in-neighbors of level 1 supporters $\{y_i\} \in B$. After one full pass, the method accumulates all unique supporters z of each node $x \in B$.

Theorem 6.1. *Assuming F is the size of the in-graph in bytes and RAM size is M, the total I/O in SUPP-A consists of read-only operations and equals*

$$D = \frac{F^2}{M}\left(1 + \frac{E[SUPP(x)]}{E[X_i]}\right). \tag{5}$$

Proof. Assuming h is the node-label hash size, let $K = |B|/(hE[X_i])$ be the number of vectors loaded in buffer B. Then $hE[SUPP(x)]K$ is the amount of RAM needed to keep the supporters for these nodes. Since memory size $M = |B| + hE[SUPP(x)]K$, we have

$$|B| = \frac{E[X_i]M}{E[X_i] + E[SUPP(x)]}. \tag{6}$$

Table 6. Disk I/O and RAM for External Memory Processing

	IRLbot (15.8GB)			ClueWeb09 (3.6GB)		
Algorithm	Read	Write	RAM	Read	Write	RAM
SUPP-A	32TB	–	8GB	fits in RAM		8GB
	130TB	–	2GB	56TB	–	2GB
	2.6PB	–	100MB	1.1PB	–	100MB
SUPP-B	49TB	49TB	8GB	fits in RAM		8GB
	98TB	98TB	2GB	136TB	136TB	2GB
	147TB	147TB	100MB	204TB	204TB	100MB
Bit Vector ($r = 2$)	63GB	–	1.9GB	15GB	–	614MB
Quick Visit	31.4GB	–	2.1GB	7.4GB	–	737MB
TSE ($p = 10^{-2}$)	31.4GB	–	157MB	7.4GB	–	37MB
TSE ($p = 10^{-3}$)	31.4GB	–	16MB	7.4GB	–	4MB
TSE ($p = 10^{-4}$)	31.4GB	–	1.6MB	7.4GB	–	0.4MB

It takes $F/|B|$ passes to finish processing the graph and the amount of disk I/O in each pass is F. Thus, the total I/O overhead is $D = F^2/|B|$. Applying Equation (6), we get Equation (5). □

Another approach, which we call SUPP-B, performs much better when M is small. It scans simultaneously both in/out-graphs (assumed to be in sorted order), obtains lists $y \leftarrow \{z_j\}$ and $y \rightarrow \{x_i\}$, and writes out all pairs (x_i, z_j), where z_j is x_i's level 2 supporter. Note that many of the pairs are duplicates, which require external memory sorting to eliminate. This method initially produces $D = 2nhE[Q(x)]$ bytes to disk, which represent $m = D/M$ sorted blocks that need to be k-way merged. Using $B = 256$KB per buffer for each open file handle, SUPP-B uses $k = M/B$ concurrent handles and $\lceil \log_k m \rceil$ merge phases.

QVS reads the file twice (i.e., $D = 2F$) and stores two vectors of in-degree counts and hashes in RAM. This gives $M = 2(h + d)n$, where $d = 4$, the number of bytes needed for each supporter counter. BV has been proposed [10] under the assumption that entire bit vectors from the last two iterations and all supporter counts fit in RAM. Its memory consumption is therefore $M = (2b + d)n$ and its disk overhead is $D = 2rF$. Finally, TSE reads $D = 2F$ bytes and maintains $M = pF$, without ever requiring that all supporter counts fit in RAM. Note that since both BV and QVS are iterative methods similar to PageRank, their M can be reduced at the expense of certain offline preprocessing on the graph [19]. We do not consider this in our comparison.

Table 6 shows the resulting I/O cost for all studied methods using PLD graphs with 8-byte hashes. We ignore the small memory needed for one or two file handles (SUPP-A, QVS, BV, TSE) and the buffer through which each graph is sequentially scanned (all methods). For the IRLbot case, SUPP-A with 8GB of RAM (i.e., half the graph fits in memory) requires 32TB of disk I/O because it reads the graph 2,074 times. As even less RAM becomes available, SUPP-A's performance worsens and reaches a pretty insane 2.6PB with $M = 100$MB. While 70 hours of CPU time in the previous section may seem extreme, this much disk activity will take much longer. For small M (e.g., 100MB in the table), SUPP-B maintains much better scalability but requires enough disk space to write 49TB of pairs (x, z) and then perform a three-pass merge with 294TB of combined read/writes. Both BV and QVS perform well, scanning the graph just 2 to 4× and maintaining approximately 2GB in RAM. However, TSE lowers their M by an additional one to three orders of magnitude depending on p. On ClueWeb09, SUPP-A issues 2.3× less I/O than on IRLbot, while SUPP-B requires 1.4× more. Either way, these results indicate that SUPP can be prohibitively expensive even on graphs 4× smaller than IRLbot's, unless they fit in RAM.

As $n \to \infty$, one last remark is that TSE may use a more aggressive p to maintain asymptotically constant RAM consumption and linear I/O, which none of the other methods admit. Combining this with earlier findings in this section, TSE represents the most accurate and scalable solution to estimating SUPP.

7 CONCLUSION

This article compared various agnostic algorithms for ranking the web at the PLD level, using both manual analysis and Google Toolbar Ranks. We showed that SUPP decisively outperformed the other methods on the two largest academic datasets. Even though the web has changed since these crawls were made, we believe the main challenges in real-time ranking and principles behind our conclusion are still in effect today. These include importance and difficulty of ranking the frontier, high cost for spammers to obtain links from many unique PLDs, and SUPP being more resilient to spam than the other algorithms at the domain level, where the top of the list requires support from millions, rather than thousands, of domains. As SUPP was infeasible in practice due to the high computational complexity, we proposed a fast, scalable, and accurate estimator called TSE for its top-ranked PLDs. It is shown to achieve a 1% error in the top 1K list with four orders of magnitude less CPU and I/O overhead than SUPP.

Future work involves incorporating TSE into a real web crawler and verifying its performance in ranking a dynamically growing PLD graph, as well as collecting new datasets to verify our conclusions.

REFERENCES

[1] Jacob Abernethy, Olivier Chapelle, and Carlos Castillo. 2008. Web spam identification through content and hyperlinks. In *Workshop on Adversarial Information Retrieval on the Web*. 41–44. DOI: http://dx.doi.org/10.1145/1451983.1451994

[2] Serge Abiteboul, Mihai Preda, and Gregory Cobena. 2003. Adaptive on-line page importance computation. In *International Conference on World Wide Web (WWW)*. 280–290.

[3] Sarker Tanzir Ahmed and Dmitri Loguinov. 2015. Around the web in six weeks: Documenting a large-scale crawl. In *IEEE Conference on Computer Communications (INFOCOM)*. 1598–1606.

[4] alexa.com. Retrieved in 2015 from http://www.alexa.com/.

[5] Jesse Alpert and Nissan Hajaj. 2008. We Knew the Web Was Big... (July 2008). Retrieved from http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

[6] Shatlyk Ashyralyyev, B. Barla Cambazoglu, and Cevdet Aykanat. 2013. Incorporating the surfing behavior of web users into pagerank. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 2351–2356.

[7] Ricardo Baeza-Yates, Carlos Castillo, Mauricio Marin, and Andrea Rodriguez. 2005. Crawling a country: Better strategies than breadth-first for web page ordering. In *International Conference on World Wide Web (WWW)*.

[8] Xiao Bai, B. Barla Cambazoglu, and Flavio P. Junqueira. 2011. Discovering URLs through user feedback. In *ACM International Conference on Information and Knowledge Management (CIKM)*. 77–86.

[9] Luca Becchetti, Carlos Castillo, Debora Donato, Stefano Leonardi, and Ricardo Baeza-Yates. 2006. Link-based characterization and detection of web spam. In *Workshop on Adversarial Information Retrieval on the Web*.

[10] Luca Becchetti, Carlos Castillo, Debora Donato, Stefano Leonardi, and Ricardo Baeza-Yates. 2006. Using rank propagation and probabilistic counting for link-based spam detection. In *Workshop on Web Mining and Web Usage Analysis (WebKDD)*.

[11] Andras Benczur, Karoly Csalogany, and Tamas Sarlos. 2006. Link-based similarity search to fight web spam. In *Workshop on Adversarial Information Retrieval on the Web*.

[12] Andras A. Benczur, Karoly Csalogany, Tamas Sarlos, and Mate Uher. 2005. Spamrank – Fully automatic link spam detection. In *Workshop on Adversarial Information Retrieval on the Web*.

[13] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. UbiCrawler: A scalable fully distributed web crawler. *Software: Practice & Experience* 34, 8 (July 2004), 711–726.

[14] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. In *International Conference on World Wide Web (WWW)*. 107–117.

[15] Carlos Castillo, Mauricio Marin, Andrea Rodriguez, and Ricardo Baeza-Yates. 2004. Scheduling algorithms for web crawling. In *Latin American Web Conference*.

[16] James Caverlee and Ling Liu. 2007. Countering web spam with credibility-based link analysis. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 157–166.
[17] James Caverlee, Steve Webb, and Ling Liu. 2007. Spam-resilient web rankings via influence throttling. In *International Parallel and Distributed Processing Symposium (IPDPS)*.
[18] Anne Chao and Shen-Ming Lee. 1992. Estimating the number of classes via sample coverage. *Journal of the American Statistical Association* 87, 417 (March 1992), 210–217.
[19] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. 2002. I/O-efficient techniques for computing pagerank. In *International Conference on Information and Knowledge Management (CIKM)*.
[20] Yen-Yu Chen, Qingqing Gan, and Torsten Suel. 2004. Local methods for estimating pagerank values. In *International Conference on Information and Knowledge Management (CIKM)*.
[21] Junghoo Cho, Hector Garcia-Molina, Taher Haveliwala, Wang Lam, Andreas Paepcke, and Sriram Raghavan Gary Wesley. 2006. Stanford webbase components and applications. *ACM Transactions on Internet Technology* 6, 2 (May 2006), 153–186.
[22] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. 1998. Efficient crawling through URL ordering. In *International Conference on World Wide Web (WWW)*. 161–172.
[23] Young-Joo Chung, Masashi Toyoda, and Masaru Kitsuregawa. 2009. A study of link farm distribution and evolution using a time series of web snapshots. In *Workshop on Adversarial Information Retrieval on the Web*. 9–16.
[24] ClueWeb09 Dataset. Retrieved in 2015 from http://www.lemurproject.org/clueweb09/.
[25] CLuE cluster. Retrieved from http://www.nsf.gov/cise/clue/index.jsp.
[26] ClueWeb09 documentation. Retrieved in 2015 from http://boston.lti.cs.cmu.edu/Data/web08-bst/planning.html.
[27] Isabel Drost and Tobias Scheffer. 2005. Thwarting the nigritude ultramaring: Learning to identify link spam. In *European Conference on Machine Learning (ECML)*.
[28] N. Duffield, C. Lund, and M. Thorup. 2003. Estimating flow distributions from sampled flow statistics. In *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 325–336.
[29] Jenny Edwards, Kevin McCurley, and John Tomlin. 2001. An adaptive model for optimizing performance of an incremental web crawler. In *International Conference on World Wide Web (WWW)*. 106–113.
[30] David Eichmann. 1994. The RBSE spider – Balancing effective search against web load. In *International Conference on World Wide Web (WWW)*.
[31] Nadav Eiron, Kevin S. McCurley, and John A. Tomlin. 2004. Ranking the web frontier. In *International Conference on World Wide Web (WWW)*. 309–318.
[32] Guang Feng, Tie-Yan Liu, Ying Wang, Ying Bao, Zhiming Ma, Xu-Dong Zhang, and Wei-Ying Ma. 2006. AggregateRank: Bringing order to web sites. In *ACM SIGIR Conference on Research and Development in Information Retrieval*. 75–82.
[33] Dennis Fetterly, Mark Manasse, and Marc Najork. 2004. Spam, damn spam, and statistics: Using statistical analysis to locate spam web pages. In *International Workshop on the Web and Databases (WebDB)*. 1–6.
[34] Google Toolbar. Retrieved in 2008 from http://toolbar.google.com.
[35] IRLbot PLD graph. Retrieved in 2018 from http://irl.cs.tamu.edu/projects/web/.
[36] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Zien. 2004. How to build a webfountain: An architecture for very large-scale text analytics. *IBM Systems Journal* 43, 1 (2004), 64–77.
[37] Zoltán Gyöngyi, Pavel Berkhin, Hector Garcia-Molina, and Jan Pedersen. 2006. Link spam detection based on mass estimation. In *International Conference on Very Large Data Bases (VLDB)*. 439–450.
[38] Zoltán Gyöngyi and Hector Garcia-Molina. 2005. Link spam alliances. In *International Conference on Very Large Data Bases (VLDB)*. 517–528.
[39] Zoltán Gyöngyi and Hector Garcia-Molina. 2005. Web spam taxonomy. In *Workshop on Adversarial Information Retrieval on the Web*. 39–47.
[40] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. 2004. Combating web spam with trustRank. In *International Conference on Very Large Data Bases (VLDB)*. 576–587.
[41] Younes Hafri and Chabane Djeraba. 2004. High performance crawling system. In *ACM International Workshop on Multimedia Information Retrieval (MIR)*. 299–306.
[42] Allan Heydon and Marc Najork. 1999. Mercator: A scalable, extensible web crawler. *World Wide Web* 2, 4 (Dec. 1999), 219–229.
[43] Jun Hirai, Sriram Raghavan, Hector Garcia-Molina, and Andreas Paepcke. 2000. WebBase: A repository of web pages. In *International Conference on World Wide Web (WWW)*. 277–293.
[44] Internet Archive. Retrieved in 2018 from http://archive.org/.
[45] Melody Ivory and Marti Hearst. 2002. Statistical profiles of highly-rated web sites. In *Conference on Human Factors in Computing Systems (CHI)*. 367–374.

[46] Kasom Koht-arsa and Surasak Sanguanpong. 2002. High performance large scale web spider architecture. In *IEEE International Symposium on Communications and Information Technologies (ISCIT)*.
[47] Laboratory for Web Algorithmics. Retrieved in 2018 from http://law.dsi.unimi.it/.
[48] Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov. 2009. IRLbot: Scaling to 6 billion pages and beyond. *ACM Transactions on the Web* 3, 3 (June 2009), 1–34.
[49] Ronny Lempel and Shlomo Moran. 2001. SALSA: The stochastic approach for link-structure analysis. *ACM Transactions on Information Systems* 19, 2 (Apr. 2001), 131–160.
[50] Yanhong Li. 1998. Toward a qualitative search engine. *IEEE Internet Computing* 2, 4 (July 1998), 24–29.
[51] Yuting Liu, Bin Gao, Tie-Yan Liu, Ying Zhang, Zhiming Ma, Shuyuan He, and Hang Li. 2008. BrowseRank: Letting web users vote for page importance. In *ACM SIGIR Conference on Research and Development in Information Retrieval*. 451–458.
[52] Oliver A. McBryan. 1994. GENVL and WWWW: Tools for taming the web. In *International Conference on World Wide Web (WWW)*.
[53] Gilad Mishne, David Carmel, and Ronny Lempel. 2005. Blocking blog spam with language model disagreement. In *Workshop on Adversarial Information Retrieval on the Web*.
[54] Mozilla. 2013. Public Suffix List. (May 2013). Retrieved from http://publicsuffix.org/.
[55] Marc Najork and Allan Heydon. 2001. *High-Performance Web Crawling*. Technical Report 173. Compaq SRC. Retrieved from http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-173.pdf.
[56] Marc Najork and Janet L. Wiener. 2001. Breadth-first search crawling yields high-quality pages. In *International Conference on World Wide Web (WWW)*. 114–118.
[57] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. 2005. Object-level ranking: Bringing order to web objects. In *International Conference on World Wide Web (WWW)*.
[58] Alexandros Ntoulas, Marc Najork, Mark Manasse, and Dennis Fetterly. 2006. Detecting spam web pages through content analysis. In *International Conference on World Wide Web (WWW)*. 83–92. DOI: http://dx.doi.org/10.1145/1135777.1135794
[59] Nutch. Retrieved in 2018 from http://nutch.apache.org/.
[60] Jothi Padmanabhan. 2010. Introduction to Grid Computing via Map-Reduce & Hadoop. (Dec. 2010). Retrieved from http://internationalnetworking.iu.edu/sites/internationalnetworking.iu.edu/files/indo-us-preso.pdf.
[61] Brian Pinkerton. 1994. Finding what people want: Experiences with the web crawler. In *International Conference on World Wide Web (WWW)*.
[62] Brian Pinkerton. 2000. *WebCrawler: Finding What People Want*. Ph.D. Dissertation. University of Washington.
[63] Jakub Piskorski, Marcin Sydow, and Dawid Weiss. 2008. Exploring linguistic features for web spam detection: A preliminary study. In *Workshop on Adversarial Information Retrieval on the Web*. 25–28. DOI: http://dx.doi.org/10.1145/1451983.1451990
[64] Matthew Richardson, Amit Prakash, and Eric Brill. 2006. Beyond Pagerank: Machine learning for static ranking. In *International Conference on World Wide Web (WWW)*. 707–715.
[65] Hiroo Saito, Masashi Toyoda, Masaru Kitsuregawa, and Kazuyuki Aihara. 2007. A large-scale study of link spam detection by graph algorithms. In *Workshop on Adversarial Information Retrieval on the Web*. 45–48.
[66] Vladislav Shkapenyuk and Torsten Suel. 2002. Design and implementation of a high-performance distributed web crawler. In *IEEE International Conference on Data Engineering (ICDE)*. 357–368.
[67] Amit Signhal. 2012. Breakfast with Google's Search Team. (Aug. 2012). Retrieved from https://www.youtube.com/watch?v=8a2VmxqFg8A.
[68] Aameek Singh, Mudhakar Srivatsa, Ling Liu, and Todd Miller. 2003. Apoidea: A decentralized peer-to-peer architecture for crawling the world wide web. In *SIGIR Workshop on Distributed Information Retrieval*. 126–142.
[69] William Stearns. 2008. SpamAssassin Domain Blacklist. (Aug. 2008). Retrieved from http://www.stearns.org/sa-blacklist/.
[70] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram. 2003. ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In *International Workshop on the Web and Databases (WebDB)*. 67–72.
[71] The Stanford WebBase Project. 2008. WebBase Archive. (Sept. 2008). Retrieved from http://dbpubs.stanford.edu:8091/~testbed/doc2/WebBase/.
[72] John Tomlin. 2003. A new paradigm for ranking pages on the world wide web. In *International Conference on World Wide Web (WWW)*.
[73] Trystan Upstill, Nick Craswell, and David Hawking. 2003. Predicting fame and fortune: PageRank or indegree. In *8th Australasian Document Computing Symposium*.
[74] X. Wang, X. Li, and D. Loguinov. 2011. Modeling residual-geometric flow sampling. In *IEEE Conference on Computer Communications (INFOCOM)*. 1808–1816.

[75] Steve Webb, James Caverlee, and Calton Pu. 2007. Characterizing web spam using content and HTTP session analysis. In *Conference on Email and Anti-Spam (CEAS)*.

[76] Steve Webb, James Caverlee, and Calton Pu. 2008. Predicting web spam with HTTP session information. In *International Conference on Information and Knowledge Management (CIKM)*. 339–348. DOI: http://dx.doi.org/10.1145/1458082.1458129

[77] Baoning Wu and Kumar Chellapilla. 2007. Extracting link spam using biased random walks from spam seed sets. In *Workshop on Adversarial Information Retrieval on the Web*. 37–44.

[78] Baoning Wu and Brian Davison. 2005. Identifying link farm spam pages. In *International Conference on World Wide Web (WWW)*.

[79] Baoning Wu, Vinay Goel, and Brian Davison. 2006. Topical trustrank: Using topicality to combat web spam. In *International Conference on World Wide Web (WWW)*. 63–72.

[80] J. Wu and K. Aberer. 2004. Using Siterank for decentralized computation of web document ranking. In *Adaptive Hypermedia*. 265–274.

[81] Gui-Rong Xue, Qiang Yang, Hua-Jun Zeng, Yong Yu, and Zheng Chen. 2005. Exploiting the hierarchical structure for link analysis. In *ACM SIGIR Conference on Research and Development in Information Retrieval*. 186–193.

[82] Sandeep Yadav, Ashwath Kumar Krishna Reddy, A. L. Narasimha Reddy, and Supranamaya Ranjan. 2010. Detecting algorithmically generated malicious domain names. In *ACM SIGCOMM Internet Measurement Conference (IMC)*. ACM, 48–61.

[83] Yahoo Research Barcelona. 2008. Datasets for Research on Web Spam Detection. (Sept. 2008). Retrieved from http://barcelona.research.yahoo.net/webspam/datasets/.

[84] Hongfei Yan, Jianyong Wang, Xiaoming Li, and Lin Guo. 2002. Architectural design and evaluation of an efficient web-crawling system. *Journal of Systems and Software* 60, 3 (Feb. 2002), 185–193.

Received October 2017; revised May 2018; accepted July 2018