Around the Web in Six Weeks: Documenting a Large-Scale Crawl

Sarker Tanzir Ahmed, Clint Sparkman†, Hsin-Tsang Lee, Dmitri Loguinov*
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843, USA
Email: {tanzir, sparkman}@cse.tamu.edu, htlee@microsoft.com, dmitri@cse.tamu.edu

***Abstract*—Exponential growth of the web continues to present challenges to the design and scalability of web crawlers. Our previous work on a high-performance platform called IRLbot [28] led to the development of new algorithms for realtime URL manipulation, domain ranking, and budgeting, which were tested in a 6.3B-page crawl. Since very little is known about the crawl itself, our goal in this paper is to undertake an extensive measurement study of the collected dataset and document its crawl dynamics. We also propose a framework for modeling the scaling rate of various data structures as crawl size goes to infinity and offer a methodology for comparing crawl coverage to that of commercial search engines.**

I. Introduction

Web crawling is not just an important component of major search engines, but also a vital experimental activity that provides indispensable research data for such fields as networking, distributed systems, databases, machine learning, information retrieval, security, and linguistics. For years, research into large-scale web crawling has been led by industry players (e.g., Google, Microsoft, Yahoo) and has remained shrouded in secrecy. While many academic crawlers have been proposed in the literature [3], [8], [14], [15], [18], [17], [19], [22], [24], [25], [27], [30], [36], [37], [41], their scale, Internet coverage, download speed, and ability to deal with spam have not kept up with the evolution of the web.

Part of the problem is the perceived impossibility, and thus lacking attempts, to rival commercial search engines using a research implementation. This notion commonly stems from a belief that to achieve meaningful results web crawlers must be heavily parallelized and even distributed across multiple domains [8], [16], [29], [31], [32], [37], [39], [40]. As a consequence, not much effort has been put into optimizing individual servers, improving their algorithms, or reducing complexity, all under the assumption that arbitrary scalability could be achieved just by acquiring "enough" hardware. Unfortunately, due to the serious financial investment needed for this vision, few academic crawlers have gone beyond small-scale prototypes.

The second problem is that prior crawls are often poorly documented and difficult to interpret. As the field stands today, there exists no standard methodology for examining web crawls and comparing their performance against one another. With each paper providing different, and often very limited, types of information, little can be said about the relative strengths of various crawling techniques or even their web coverage. Setting aside the financial aspect discussed above, this lack of transparency has helped stifle innovation, allowing industry to take a technological and scientific lead in this area.

Finally, the majority of existing web studies have no provisions to handle spam [8], [12], [17], [30], [36], [37], [41]. One technique for tackling the massive scale, infinite script-generated traps, and uncertainty about content quality is to select a handful of "good" sites and then crawl them until some maximum number of pages is reached within each host [8], [17]. While suitable in some cases, this approach does not easily generalize to Internet-wide crawling scenarios.

To overcome these limitations, our contribution in this paper is to propose a new methodology for understanding web crawls, set forth guidelines for systematically analyzing crawler performance, provide evidence that high-performance web exploration is possible using a research implementation, and dissect the main IRLbot experiment that still remains the largest and fastest endeavor in the literature. It is also the only documented crawler that used real-time prioritization of downloaded pages in an effort to avoid spam. Our additional contribution is to analyze the growth rate of various data structures as crawl size scales up and introduce techniques for assessing web coverage using commercial search engines, neither of which has been attempted before.

II. Understanding Web Crawls

This section examines the various challenges in web crawling, identifies metrics of interest, and reviews prior work.

A. Crawler Operation

As shown in Fig. 1, crawler operation can be reduced to a cycle with eight major components. Besides maintaining many concurrent HTTP sessions and parsing HTML, crawlers must eliminate duplicate URLs before attempting to crawl them, rank the frontier (i.e., decide importance of all seen, but not-yet-crawled pages), perform admission control on pending URLs using ranks computed in real-time, execute DNS lookups, enforce `robots.txt` directives of crawled

†Major Clint Sparkman is an active duty member of the United States Air Force. The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

*Supported by NSF grants CNS-1017766 and CNS-1319984.

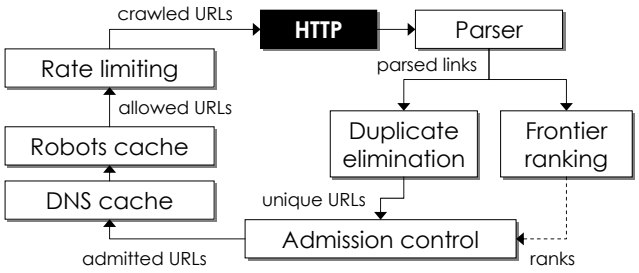


Fig. 1. Components of modern crawlers.

websites, and finally adhere to politeness rate limits (both per-IP and per-host) that prevent crashing of individual servers.

The ability of each component to keep up with the crawl is determined by two factors – size of the underlying data structures and speed at which they must operate. Suppose $\mathcal{D}$ is the number of downloaded pages, q is the fraction of them with error-free (i.e., 200 OK) HTML content, and h is the number of crawling servers. Further assume that the parser produces on average l links per page that are *locally* unique (i.e., within that page). After verification against all previously seen URLs, suppose fraction p of them are *globally* unique.

Armed with these definitions, we can now ballpark the storage and processing demand of each crawling server. Duplicate elimination uses a data structure with $\mathcal{D}qlp/h$ hashes of seen pages, admission control keeps track of all unique pages minus those already crawled, which amounts to $\mathcal{D}q(lp-1)/h$ full URL strings, and frontier ranking operates on webgraphs with $\mathcal{D}ql/h$ edges. The overhead of the other three components is dependent on the number of hosts visited by the crawler, cache expiration delays (DNS and robots), and the desired website concurrency in the politeness scheduler.[1]

Components closer to the parser in the clockwise direction need to support higher throughput rates. For a target speed of $\mathcal{S}$ crawled pages/s (pps), both duplicate elimination and frontier ranking must operate at $\mathcal{S}ql/h$ links/s (lps), while admission control must cope with an injection rate $\mathcal{S}qlp/h$ URLs/s and extraction rate $\mathcal{S}/h$, the latter of which is also the speed of the remaining components in the figure. Given $l \approx 50$, $h = 1$, and peak rate $\mathcal{S} = 3$K pps from IRLbot experiments, verification of uniqueness and injection of edges into the webgraph must proceed at rates well in excess of 100K/s.

As larger data structures require more processing, crawler design boils down to a tradeoff between four parameters $\{\mathcal{D}/h, \mathcal{S}/h, q, l\}$, where increase in one parameter typically requires reduction in other parameters. However, this also leads to crawls with incompatible results and difficulties in gauging performance of their underlying algorithms. We discuss several examples next.

B. Duplicate Elimination

Due to the tiny size of the early web, first-generation crawlers [9], [23], [30], [36] either kept all data in RAM or used random disk access to verify URL uniqueness. With 8-ms seek delays and worst-case access locality, these methods in today's Internet would bottleneck around $\mathcal{S} = 125/l \approx 2.5$ pps. Second-generation crawlers [32], [37] replaced disk seeking with batch-mode sorting that periodically scanned the file of previously seen URLs and merged the new ones in. While this approach works well for a few hundred million pages, scaling it further requires a proportionate reduction in $\mathcal{S}$, in some cases pushing the crawler to a virtual standstill [28].

Subsequent literature, which we call third-generation, universally dismissed single-server designs and assumed that scalability was only achievable horizontally, i.e., by increasing h. This work [16], [29], [31], [39], [40], [41] focused on parallelizing the URL workload across server clusters and P2P networks. However, even with distributed operation, experiments with these designs lasted only minutes and produces crawls limited to 400K-25M pages, with no measurable improvement in the last decade.

Besides trading $\mathcal{S}$ for $\mathcal{D}$ and scaling h, other methods include reduction in q (i.e., download of non-HTML objects) [23], [33], elimination of dynamic links (e.g., forums, blogs, social networks, shopping sites) to reduce l [20], and avoidance of disk-based uniqueness verification altogether by either keeping all data structures in RAM [5], [8], [17] or revisiting the same pages on a regular basis [12], [24], [25], [26].

C. Ranking and Admission Control

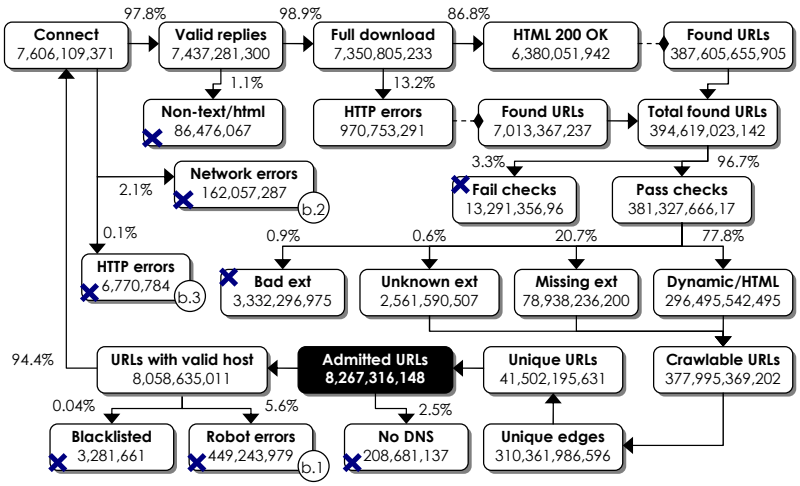
Several papers [3], [15], [6], [14], [18], [19], [22], [27] have proposed that crawlers compute a certain graph-theoretic metric for each page (e.g., PageRank [9], OPIC [1]) and that pending URLs be served in the order of their rank. However, due to the high CPU and I/O cost, most of these efforts remain limited to offline simulations. Among the crawls in the literature with at least 50M pages [8], [17], [23], [32], [37], none have used real-time spam avoidance or global frontier prioritization, most often relying on variations of polite BFS to automatically find good pages [33].

While open-source implementations exist with non-BFS capability [35], they do not publish performance results or disclose operational details, which makes their analysis difficult. They also often require substantial resources (e.g., clusters with large h, terabytes of RAM) and lower l to sustain non-trivial crawls. One notable example is ClueWeb09 [20], which parallelized Apache Nutch [35] using the NSF-Google-IBM cluster with 1,600 processors [34]. After discarding all dynamic links (i.e., dropping l by 84%), the experiment finished 1B pages in 52 days at an average rate of 222 pps; however, little additional detail is available about this dataset, its crawl dynamics, web coverage, or the employed algorithms.

D. Discussion

For future web-crawling research to provide credible results, we believe that one should engage in experimentation that aims to simultaneously surpass prior crawls in all four metrics introduced earlier in this section – $\mathcal{S}/h$, $\mathcal{D}/h$, q, l. Besides scalability, it is also important to consider the

[1]For example, IRLbot began stalling if the number of websites with backlogged (i.e., allowed, but not yet crawled) URLs dropped below 200K.



(a) URL cycle

Category	Error type	URLs affected
(b.1) Robots	Robots.txt disallow	296,966,591
	Network error on robots.txt	106,638,856
	HTTP error on robots.txt	24,221,293
	Robots.txt forbidden	20,621,185
	Robots.txt loop	612,160
	Robots.txt over 64KB	183,894
	Total	**449,243,979**
(b.2) Network	Connect fail	124,752,717
	Receive fail	36,534,906
	Slow download	421,427
	Page over 4MB	338,872
	Send fail	9,365
	Total	**162,057,287**
(b.3) HTTP	Bad HTTP response	4,139,148
	Decompression fail	1,110,272
	Bad HTTP status	682,613
	Invalid base URL	593,941
	Bad chunking	242,858
	Header over 64 KB	1,952
	Total	**6,770,784**

(b) discarded URLs

Fig. 2. IRLbot crawl analysis.

average crawl depth $\log_{lp}(\mathcal{D}/m)$, which determines how far the crawler ventures from the m seed pages and how likely it is to encounter spam, and several auxiliary parameters – the number of crawled hosts, domains, and IPs – since they control the size of the various caches, DNS and `robots.txt` workload, complexity of politeness rate-limiting, and Internet coverage. Another relevant metric is the average page size, which together with $\mathcal{S}$ determines the download bandwidth and performance of the network stack.

As we show in the remainder of the paper, IRLbot is an implementation of this vision using $m = h = 1$, maximum q, and unrestricted l, with $\mathcal{S}$ and $\mathcal{D}$ determined by forces outside our control (i.e., university bandwidth).

III. Page-Level Analysis

We next explain our proposed methodology for documenting large-scale crawls. This section underscores the importance of collecting extensive statistics and meticulously logging the various failure conditions, many of which are routinely omitted from prior studies.

A. Admitted URLs

To address the vagueness of prior URL statistics, we propose that they be presented under a unified umbrella, which we call the *URL cycle*. Fig. 2 shows its basic structure. Over a period of six weeks, IRLbot pulled $\mathcal{A} = 8.2$B URLs from admission control (the shaded box in part (a) of the figure) and attempted to crawl them. Approximately 2.5% of these pointed to hosts without DNS entries and were immediately discarded. Out of the remaining 8B links, 0.04% were thrown out due to manual blacklisting in response to complaints and an additional 5.6% were dropped during the `robots.txt` phase. Part (b.1) of the figure shows a detailed breakdown of these errors.

While IRLbot attempted redirects on `robots.txt` back-to-back, normal URLs were handled differently. To avoid wasting bandwidth on spam that frequently employed lengthy sequences of redirects, IRLbot treated each $301/302$ as a new link (i.e., sent it for regular uniqueness verification and then admission control). This ensured that redirects had to pass spam-related budget enforcement before being attempted again, which made the retry latency dependent on the current rank of the corresponding domain and its URL backlog.

B. Crawled URLs

After passing the robot phase, $\mathcal{C} = 7.6$B URLs continued through the cycle and were issued non-robot connection requests. This resulted in 162M network errors and 6.7M HTTP failure conditions. The breakdown of the former is shown in part (b.2) of the figure, where the most common reasons were connect (124M) and receive (36M) failures. One peculiar category of (b.2) consists of 421K URLs that were aborted when the host either did not provide any data for over 60 seconds or dragged out the download beyond 180 seconds, which were common spammer tactics aimed at stalling IRLbot. The opposite technique was to serve "infinite" streams of data, which we terminated at 4 MB, resulting in 338K additional pages being discarded. Despite this limit, the largest document the parser dealt with (after decompression) was 884 MB.

The most common HTTP failure in (b.3) was the missing status line in the response, which affected 4.1M URLs. Sometimes attributed to ancient HTTP/0.9 servers, this condition might also be indicative of other services running on the contacted port and various firewall/IDS misconfigurations. The second most common error type in (b.3) was failed decompression (1.1M), with gzip corruption responsible for 1M URLs and unknown/bogus encoding type for the other 0.1M cases. Finally, 682K URLs in (b.3) had an invalid HTTP status code (i.e., above 505 or below 100), 593K contained an unparsable base URL, 242K violated the chunking syntax or exceeded 4 MB after unchunking, and 1.9K contained HTTP headers over 64 KB.

Going back to Fig. 2(a), $\mathcal{R} = 7.4$B valid replies produced $\mathcal{O} = 86.5$M objects with status 200 OK and content-type other

TABLE I
HTTP STATUS CODES OF DOWNLOADED URLS

	IRLbot [28]	Mercator [23], [32]		Polybot [37]
$\mathcal{D}$	7.3B	76M	819M	139M
$\mathcal{H}$	6.3B	45M	473M	unknown
HTTP 200	86.79%	87.03%	88.50%	87.42%
302	7.49%	3.33%	3.31%	4.37%
404	3.56%	7.43%	6.46%	5.32%
301	1.12%	1.12%	–	2.08%
500	0.35%	0.11%	–	0.07%
403	0.28%	0.43%	–	0.35%
400	0.10%	0.09%	–	–
401	0.09%	0.30%	–	0.30%
406	0.08%	0.11%	–	–
Other	0.12%	0.06%	1.73%	0.09%

than text/html. To build as massive a webgraph as possible and push IRLbot to its scalability limits, we were only interested in downloading HTML pages. Considering that certain non-HTML files were extremely large (e.g., DVDs, ISOs), IRLbot aborted their connections as soon as the HTTP header was received. This curtailed the download to an average of 8.3 KB per object and limited the total wasted bandwidth to just 718 GB. Without header peeking, the crawler would have had to fall back on the 4-MB maximum page size, which could have allowed these 86.5M objects to consume 346 TB in the worst case.

C. Downloaded URLs

For the remaining $\mathcal{D} = 7.3$B URLs, 60% of which were dynamic (i.e., contained a ?), the HTTP response was fully downloaded by IRLbot. The breakdown of HTTP status codes among this group is shown in Table I, including similar statistics from three other Internet-wide crawls that supply this information. Successful pages (200 OK) accounted for $\mathcal{H} = 6.3$B responses, or approximately $q = 87\%$ of $\mathcal{D}$, and errors for $\mathcal{E} = 970$M pages. Interestingly, half of IRLbot's 200 OK pages (i.e., 3.2B) and 47% of errors (i.e., 456M) were chunked by the server, indicating some type of dynamically assembled content.

To avoid pulling non-html pages, IRLbot transmitted the "Accept: text/html" header with all non-robot.txt requests, which the server should reject with 406 Not Acceptable if the MIME type does not match the one requested by the client. Combining Table I and Fig. 2 we obtain that IRLbot attempted to download 92.6M non-HTML pages, out of which 6.1M returned with status code 406 and the remaining $\mathcal{O} = 86.5$M with 200 OK. This shows that Internet servers universally ignore the Accept field and that its usage amounts to a disappointing 6.6% reduction in aborted pages.

D. Links

Parsing a-href, frame-src, and meta-refresh tags, as well as HTTP "Location:" fields, in the 7.3B downloaded responses, IRLbot produced a total of $\mathcal{K}_1 = 394$B links shown in Fig. 2(a), with 1.7% coming from non-200 pages. The most prolific HTML page contained 4.6M links and the most verbose error page 110K. Note that unlike some of the prior work [32], we completely ignored `img` tags and did not consider them part of the URL cycle.

To avoid hitting obviously bogus pages, IRLbot tested links for correctness of syntax and discarded 3.3% of them due to invalid syntax or excessive length, where anything longer than 1.2 KB was considered unreasonable. Out of $\mathcal{K}_2 = 381$B remaining links, 3.3B pointed to a static page with one of 694 prohibited non-HTML extensions (e.g., office files, music/video). Interestingly, usage of a pretty extensive blacklist reduced the URL workload by only $\epsilon = 0.9\%$ and the number of aborted pages by an estimated $\epsilon\mathcal{R} = 64$M. Given 8.3 KB per aborted page, this translates into 534 GB of saved bandwidth, or a mere 0.37% of the total. This number seems small enough that in future crawls it might be simpler to drop extension filtering and handle *all* non-HTML objects in the download phase.

Returning to Fig. 2(a), the remaining URLs were considered suitable for crawling, which included 2.5B static links with an unknown extension, 78B static links without an extension, and 296B links that were either dynamic or indicative of HTML. Condensing 377B crawlable links by removing same-page duplicates and replacing URLs with 64-bit hashes, IRLbot constructed a 3-TB webgraph with $\mathcal{K} = 310$B edges and $\mathcal{U} = 41$B unique nodes, the latter of which was fed into admission control, completing the cycle in Fig. 2(a).

Treating links found in error pages as integral byproduct of crawling, it can be estimated that each good HTML page injected $\mathcal{K}_1/\mathcal{H} = 61.7$ URLs into the system, with $\mathcal{K}_2/\mathcal{H} = 59.7$ of them valid. However, only $l = \mathcal{K}/\mathcal{H} = 48.6$ were locally unique. We can now determine the probability that a locally unique link is globally unique, i.e., $p = \mathcal{U}/\mathcal{K} = 0.13$, and the number of URLs injected into admission control per crawled page, i.e., $lp = \mathcal{U}/\mathcal{H} = 6.58$. This allows us to estimate IRLbot's average crawl depth as $\log_{lp}(\mathcal{D}/m) = 12$, where $m = 1$ is the number of seed nodes. For comparison, the same metric in ClueWeb09 [13], [20], with its $m = 33$M and $\mathcal{D} = 1$B, is only 1.8.

IV. SERVER-LEVEL ANALYSIS

We now deal with network statistics. While prior crawls provide some limited host-related information, there is almost no discussion of the various IP-level interaction, experienced errors, robot downloads, or consumed bandwidth. We use IRLbot data to present our approach for streamlining this type of exposition.

A. DNS and Robots

Our first topic is crawler interaction with remote hosts and their authoritative DNS servers. We present the proposed model of this breakdown in Fig. 3. IRLbot's 41B discovered URLs belonged to 89M *pay-level domains* (PLDs), such as `google.com` or `amazon.co.uk`, and 641M sites. To avoid unnecessary overhead, IRLbot issued DNS queries only for URLs that passed the budget enforcer. This resulted in 297M DNS queries for 260M unique hosts, where repetition was caused by expiration of previously pulled records.

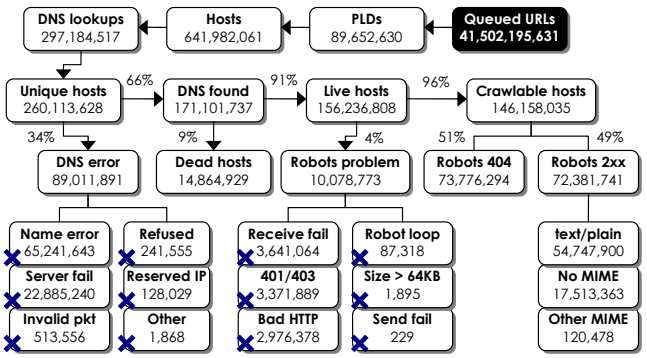


Fig. 3. DNS and robots.txt requests.

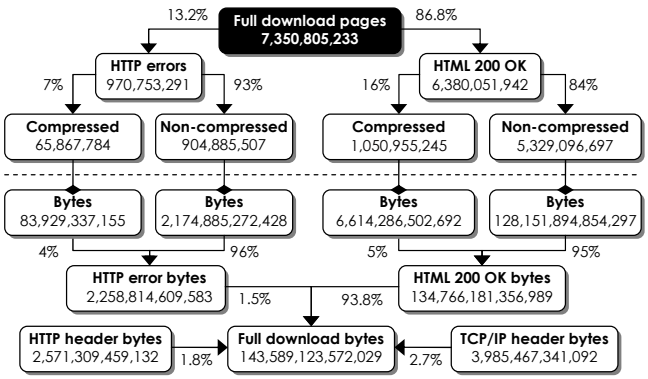


Fig. 4. Downloaded HTML pages.

Interestingly, only 171M hosts (66%) had a valid DNS entry and even fewer (156M) were live during the attempted downloaded of `robots.txt`. Among the sites that responded, 10M failed to provide a legitimate `robots.txt`, which prevented IRLbot from knowing which parts of the website should be excluded and resulted in the entire host being treated as non-crawlable. Among the remaining 146M hosts in the figure, approximately half (73M) did not use `robots.txt` and one-third (54M) served it with the correct `text/plain` MIME type. Additionally, 17M hosts provided robot files with no content-type and 120K used a non-`text/plain` type. The last two cases were often seen on servers that sent HTTP filler (e.g., custom error messages, redirects to default pages, ads) instead of proper errors, which we interpreted as equivalent to not having any crawling restrictions.

The 54M legitimate `robots.txt` files originally occupied 115 GB (i.e., 2.2 KB on average); however, after retaining only the directives that applied to either all crawlers or IRLbot specifically, the entire dataset shrunk to 1.5 GB (i.e., 28 bytes/host) and contained just 101M entries (i.e., 1.85 entries/host). This indicates that the entire collection can be easily cached in RAM, rather than on disk as done by IRLbot.

In terms of Internet coverage, 171M sites with a valid DNS entry mapped to 5,517,743 unique IPs, all of which were probed by IRLbot during attempted downloads of `robots.txt`. However, a more balanced characterization of a crawl includes only hosts with 200-OK HTML content. In that case, analysis shows that $\mathcal{H} = 6.3$B pages resided on 117,576,295 sites, 33,755,361 PLDs, and 4,260,532 IPs.

B. Bandwidth

In the outbound direction, IRLbot transmitted approximately 23 GB of DNS traffic and 33 GB of robot requests. GET packets consumed an additional 1.8 TB in HTTP headers and 1.1 TB in TCP/IP overhead. Inbound bandwidth was split across 37 GB of DNS responses, 254 GB of `robots.txt` files (including repeated requests), 718 GB of aborted objects, and 143 TB of fully downloaded URLs. A breakdown of the last category is shown in Fig. 4.

Starting with 7.3B HTML pages on top of the figure and following the path on the right, observe that 16% of all 200-OK pages arrived compressed and accounted for 6.6 TB (i.e., 6.3 KB/page). After decoding, they ballooned to a hefty 33 TB (i.e., 31.5 KB/page), showing an almost exact $5:1$ compression ratio. The other 5.3B OK pages arrived uncompressed and consumed 128 TB, as also shown in the figure. These pages were quite a bit smaller, averaging 24 KB. Not including HTTP headers, both types of 200 OK pages consumed 134 TB of bandwidth, or 93.8% of the total.

HTTP errors on the left side of Fig. 4 experienced half the likelihood of being compressed, slightly lower deflate ratios ($4.7:1$), and significantly smaller average page size, which ranged from 1.3 KB for compressed to 2.4 KB for uncompressed responses. Interestingly, error pages did not just end with the HTTP header; instead, many servers were compelled to stuff additional data after the header, including bizarre cases when it was impossible for the browser to display them. Even with this extra content, 970M HTTP errors accounted for only 1.5% of the final traffic. TCP/IP overhead (4 TB) and HTTP headers across all responses (2.5 TB) were actually more prominent.

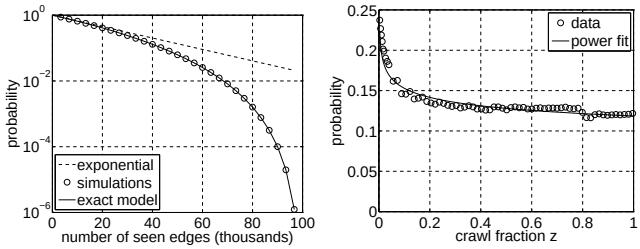
During the 41 days of the experiment, IRLbot averaged 2,132 attempted connections/s, $\mathcal{S} = 2{,}061$ full downloads/s, $\mathcal{S}q = 1{,}789$ error-free HTML pages/s, and 320 Mbps of inbound and 7 Mbps of outbound bandwidth.

V. Extrapolating Crawls

Given the large number of documents crawled by IRLbot, and even more discovered, one may wonder about the growth of the various datasets and their finiteness as the crawl continues beyond the already-seen portions of the web. We next examine this question in more detail.

A. Stochastic Model

Given a webgraph $G = (V, E)$ of the entire Internet, any crawl can be viewed as a stochastic process $\{(X_n, Y_n)\}$, where $n = 1, 2, \ldots$ is discrete time, $X_n \in V$ is the crawled page that generated link n, and $Y_n \in V$ is the URL it points to. We assume this process excludes invalid URLs, ignores same-page duplicates, and terminates after finding $N \leq |E|$ links. As the crawl progresses, we are interested in the behavior of discovery rates for new URLs, hosts, and/or PLDs. To cover all of these under a common umbrella, define indicator variable Q_n to be



(a) toy graph (b) IRLbot ($\alpha = 0.12$)

Fig. 5. Model verification for $p(t)$ and URL discovery rate $\tilde{p}(z)$ in IRLbot.

1 if link (X_n, Y_n) satisfies some uniqueness condition and 0 otherwise. For example,

$$Q_n = \begin{cases} 1 & Y_n \text{ not seen before} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

defines a non-stationary stochastic process of URL uniqueness. Then, the expected number of links L_N satisfying this condition in a crawl of size N is:

$$E[L_N] = \sum_{n=1}^{N} E[Q_n] \approx \int_1^N p(t)dt \quad (2)$$

where N is assumed to be very large and $p(t) = P(Q_t = 1) = dL_t/dt$ is the growth rate of unique nodes at time t.

It is normally expected that $p(t)$ starts off high for small t; however, as the crawler starts exhausting the web, L_t should begin experiencing saturation and thus $p(t)$ should eventually decay to zero. In this regard, two questions are in order. First, what general model does $p(t)$ follow? Second, can one estimate the number of crawled pages C_N that produce a given value L_N? For example, Google reported in 2008 reaching 1T unique nodes in the webgraph [4]. How many more pages does IRLbot have to crawl to hit the same target?

To build intuition, we answer the first question using a toy model of the web and predicate function (1). Assume the web is a finite digraph with a constant in/out-degree d and suppose the crawl implements a uniformly random shuffle on E. Then, (X_t, Y_t) points to a unique page if and only if none of Y_t's other $d-1$ in-links has been discovered in $[1, t-1]$, i.e.,

$$p(t) = \left(1 - \frac{t-1}{|E|}\right)^{d-1}. \quad (3)$$

For $t \ll |E|$, which is a common operating range of interest, Taylor expansion reduces (3) to $e^{-\lambda t}$, where $\lambda = (d-1)/|E|$. Fig. 5(a) shows simulations in comparison to (3) in a random graph with $|E| = 100$K edges and $d = 5$ (i.e., $|V| = 20$K nodes). It thus can be expected that in some cases $p(t)$ may exhibit an exponential tail, although real graphs are typically more complicated and require modeling work beyond the scope of this paper. See [2] for more detail.

We next provide a methodology to first build and then extrapolate curves $p(t)$ using real crawls. This will answer both questions posed earlier in a more realistic setting.

B. Data Extraction

We start by introducing a MapReduce algorithm for estimating $p(t)$ in a discrete set of points $t_1, t_2, \ldots, t_k$ using a given crawl dataset. Our discussion centers on URL uniqueness, but almost identical procedures apply in other cases (e.g., hosts, PLDs). Note that points $\{t_i\}$ may be spaced non-uniformly (e.g., at exponentially increasing distances), depending on the desired parameters of the plot. Define bin $b_i = [t_i - \Delta, t_i + \Delta]$ to be some Δ-neighborhood of t_i that contains enough discovered links for the law of large numbers to hold.

Assume that page-download timestamps $\tau_1, \tau_2, \ldots$ are embedded in the trace file with each crawled node. For every link (j, k), found in page j, we first determine bin i into which timestamp τ_j falls and increment the corresponding number of *seen* out-links s_i for that bin. Since the number of bins is usually small (e.g., $50 - 100$), these counters can be kept in RAM. We then map (j, k) to a tuple consisting of k's hash h_k and the crawl timestamp of the source page j, i.e., (h_k, τ_j). After all tuples are sorted by h_k, the reduce step retains the smallest timestamp for each seen URL, i.e., $(h_k, \tau_j^1, \tau_j^2, \ldots) \rightarrow (h_k, \min(\tau_j^1, \tau_j^2, \ldots))$. Scanning the final result, we obtain the number of globally unique links u_i discovered in each bin i, which produces $p(t_i) \approx u_i/s_i$.

This computation on the IRLbot dataset requires sorting 310B tuples (i.e., 3.7 TB assuming 4-byte timestamps) and produces 41B tuples (i.e., 492 GB) as output. While none of this fits in RAM, IRLbot uses disk-based algorithms that tackle this problem using a single host in a few hours.

C. URLs

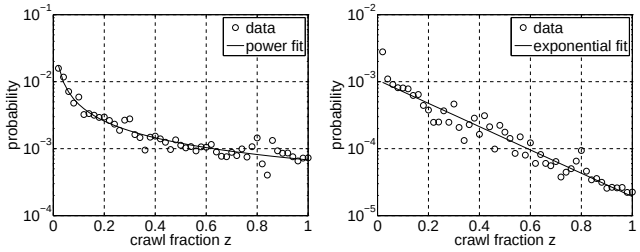
Assume $\mathcal{K}$ is the number of links in the *already-crawled* portion of the web. Then, let $z = t/\mathcal{K}$ be time normalized to this crawl and $\tilde{p}(z) = p(z\mathcal{K})$ be the corresponding uniqueness function. Using equally spaced bins and $\mathcal{K} = 310$B, Fig. 5(b) shows that IRLbot's $\tilde{p}(z)$ is a close fit to a power-law function $\beta z^{-\alpha}$, where $\alpha = 0.12$ and $\beta = 0.11$. Interestingly, this decay rate is significantly slower than predicted by (3), indicating that the degree distribution of G and crawl order (e.g., bias towards popular nodes) have a noticeable impact on the resulting curve.

We can now offer a crude model for estimating the number of crawled pages C_N at which IRLbot would hit Google's $L_N = 1$T unique URLs. To do this, we must first determine the number of links N needed to generate L_N globally unique nodes. Defining $r = N/\mathcal{K}$ and re-writing (2) in terms of normalized time, we get:

$$E[L_N] \approx \mathcal{K} \int_0^r \tilde{p}(z)dz, \quad (4)$$

where the lower limit of the integral $1/\mathcal{K}$ is approximated with a zero. As we specifically aim for cases with $r > 1$, we can split the integral into two segments, i.e., with $z \in [0, 1]$ representing the already-crawled pages and $z \in [1, r]$ being the extrapolated portion:

$$E[L_N] \approx \mathcal{U} + \mathcal{K} \int_1^r \tilde{p}(z)dz, \quad (5)$$



(a) host ($\alpha = 0.79$) (b) PLD ($\lambda = 4$)

Fig. 6. Host/PLD discovery rate $\tilde{p}(z)$ in IRLbot.

where $\mathcal{U} = 41$B is the number of unique nodes in the crawled dataset. For the Pareto tail $\tilde{p}(z) = \beta z^{-\alpha}$:

$$E[L_N] \approx \mathcal{U} + \frac{\mathcal{K}\beta(r^{1-\alpha} - 1)}{1 - \alpha}, \tag{6}$$

which in turn leads to:

$$r \approx \Big(1 + \frac{(1-\alpha)(E[L_N] - \mathcal{U})}{\mathcal{K}\beta}\Big)^{\frac{1}{1-\alpha}}. \tag{7}$$

For $\tilde{p}(z)$ in Fig. 5(b) and $E[L_N] = 1$T, this model suggests $r = 40$ times more discovered edges, which produces $N = r\mathcal{K} = 12$T links in the webgraph and $C_N = N/l = 256$B crawled pages. For 30T unique nodes seen by Google in 2012 [38], we obtain $r = 1{,}918$, $N = 592$T links, and $C_N = 12$T crawled pages. Using Google's 20B pages/day crawl rate [38], this amounts to 50 months of crawling @ 41 Gbps.

The tail of $\tilde{p}(z)$ in Fig. 5(b) is heavy enough, i.e., $\alpha < 1$, that its integral becomes unbounded as the number of seen links $N \to \infty$. While this result was anticipated knowing that scripts could generate arbitrary amounts of unique URLs, the model confirms that IRLbot was on track to experience this problem firsthand and estimates the expected growth rate of $E[L_N]$ in (6) as $\Theta(N^{0.88})$. This again cautions against attempting to download every possible unique URL and underscores the importance of prioritizing the frontier.

D. Hosts and PLDs

Applying the same methodology to the host graph, we obtain curve $\tilde{p}(z)$ in Fig. 6(a). While it drops more dramatically over the same range (i.e., by a factor of 15 instead of just 2), it still follows a power-law function, where now $\alpha = 0.79$ and $\beta = 0.0008$. Invoking (6) with $r = 40$ and $\mathcal{U} = 641$M unique nodes in the crawled portion leads to 2B extrapolated hosts, which is only 3 times larger than seen by IRLbot so far. For $r = 1{,}918$, this number scales up to 5.2B, but still remains quite reasonable. As before, $\alpha < 1$ predicts an infinite number of hosts, but their growth rate $\Theta(N^{0.21})$ is significantly slower than for unique URLs. Understanding this scaling behavior is quite useful in future designs of site-related data structures, their processing algorithms (e.g., ranking, DNS caching), and storage provisioning.

In contrast to the previous two curves, the PLD-uniqueness probability in Fig. 6(b) exhibits a different shape with a much more aggressive decay, dropping by a factor of 122 over the course of the crawl. A curve-fit suggests an exponential tail $\mu e^{-\lambda z}$ with $\lambda = 4$ and $\mu = 0.0011$, which reminds of the approximate toy model considered earlier in this section, but with an extra parameter μ. Solving the integral in (5) leads to:

$$E[L_N] \approx \mathcal{U} + \frac{\mathcal{K}\mu}{\lambda}(e^{-\lambda} - e^{-\lambda r}), \tag{8}$$

which produces $E[L_\infty] - \mathcal{U} = 1.6$M PLDs in addition to the 89M already discovered, regardless of future crawl size. IRLbot's spam avoidance has relied precisely on this fact, i.e., that spammers could not obtain control over an infinite number of PLDs.

VI. Internet-Wide Coverage

We next examine how to quantify crawl coverage of the available web space.

A. Basic Properties

Crawl coverage may be measured by 1) the collection of URLs from which 200-OK HTML content was obtained; or 2) the constructed graph of the visible web. Note that the latter consists of the former combined with the nodes in the frontier, HTTP errors, and the links connecting them together. We include errors in the webgraph as they provide valuable information about redirects (301/302), dead nodes (404/50x), forbidden URLs (401/403), and parents of crawled pages. These might be useful for merging duplicate pages, spam detection, general page ranking, and back-tracing the crawl tree, which may pass through errors, in case of complaints.

Table II shows a snapshot of available information about the major crawls in the literature. Interestingly, some papers discuss only the crawled pages (e.g., [37]), others only the web graph (e.g., [7], [10]), while some do both (e.g., [11], [21]), but often using a small subset of the possible metrics of interest. Comparison is further complicated by the various missing information and unspoken assumptions. For example, Mercator includes `img` tags in the webgraph, while other crawlers typically do not. WebBase considers HTTP errors and robot files as *crawled* pages, while others usually omit `robots.txt` from the totals and include errors only in the webgraph. UbiCrawler removes the frontier and all dangling nodes (i.e., with zero out-degree) from the webgraph, while other datasets normally retain them.

The bottom line is that accurate comparison of previous crawls is difficult; however, given the exhaustive level of detail provided earlier in this paper, IRLbot results should be straightforward to interpret.

B. TLD Coverage

Besides the raw totals in Table II, another important aspect of Internet-scale crawling is allocation of budgets to individual domains. Since no prior methods have been developed for measuring this and given that comparison of different crawls has been largely limited to graph-theoretic metrics of the webgraph (e.g., size of various bow-tie components [10], [42]), our aim in this section is to develop a novel approach for

TABLE II
INTERNET COVERAGE OF EXISTING CRAWLS

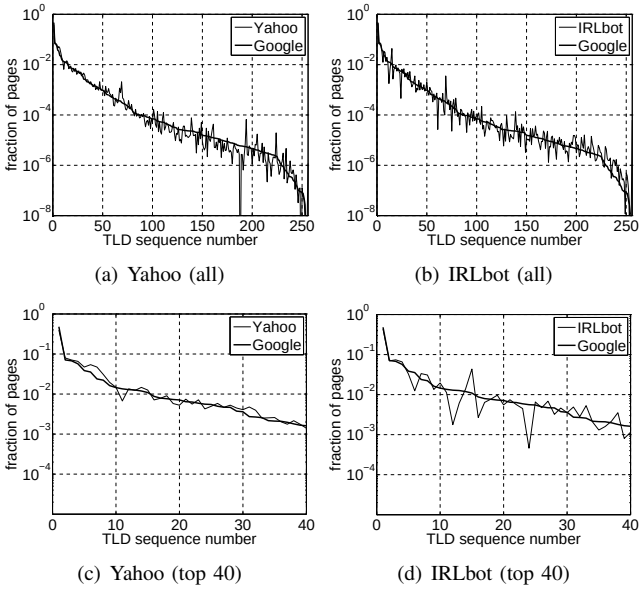
Dataset	Date	Crawled (HTML 200 OK)				Web graph		Host graph		PLD graph		TLD graph	
		pages	hosts	PLDs	TLDs	nodes	edges	nodes	edges	nodes	edges	nodes	edges
AltaVista [10]	10/99	–	–	–	–	271M	2.1B	–	–	–	–	–	–
Polybot [37]	5/01	121M	5M	–	–	–	–	–	–	–	–	–	–
Google [7]	6/01	–	–	–	–	1.3B	19.5B	12.8M	395M	–	–	–	–
Mercator [11]	7/02	429M	$\sim$ 10M	–	–	–	18.3B	–	–	–	–	–	–
WebFountain [21]	2004	1B	–	–	–	4.75B	37B	19.7M	1.1B	–	–	–	–
WebBase [17]	6/07	98M	51K	–	–	–	4.2B	–	–	–	–	–	–
ClueWeb09 [20]	1/09	1B	–	–	–	4.8B	7.9B	–	–	–	–	–	–
IRLbot	6/07	6.3B	117M	33M	256	41B	310B	641M	6.8B	89M	1.8B	256	46K
UbiCrawler .uk [8]	5/07	105M	114K	–	1	105M	3.7B	114K	–	–	–	1	1
IRLbot .uk	6/07	197M	2.8M	1.2M	1	1.3B	9.5B	5M	54M	1.5M	18M	1	1
TeaPot .cn [42]	1/06	837M	16.9M	790K	1	837M	43B	16.9M	–	790K	–	1	1
IRLbot .cn	6/07	209M	3.3M	539K	1	1.1B	11.9B	8.4M	103M	711K	19.7M	1	1

TABLE III
COMMERCIAL DATASETS

Dataset	Date	Pages
Google	1/2008	30,756,383,801
Yahoo	1/2008	37,864,090,287

TABLE IV
GOOGLE-ORDERED TOP-10 TLD LIST

TLD	Google	Yahoo	IRLbot	WebBase	ClueWeb
.com	46.7%	38.3%	43.3%	31.2%	54.8%
.net	6.9%	7.7%	6.9%	2.2%	6.7%
.de	6.6%	6.8%	7.4%	3.8%	3.8%
.org	5.5%	6.3%	6.6%	17.8%	6.6%
.cn	3.7%	4.6%	3.3%	0.2%	5.6%
.jp	3.4%	5.2%	1.2%	1.7%	3.2%
.ru	2.3%	4.6%	3.3%	0.6%	0.1%
.uk	2.2%	3.0%	3.1%	4.9%	1.7%
.pl	1.6%	1.9%	1.3%	0.2%	0.3%
.nl	1.4%	1.4%	2.0%	0.5%	0.1%
TLDs	255	256	256	174	254



(a) Yahoo (all) (b) IRLbot (all)

(c) Yahoo (top 40) (d) IRLbot (top 40)

Fig. 7. TLD coverage (Google order).

understanding how much of crawler bandwidth is spent in what parts of the Internet.

We leverage *site queries* (i.e., strings in the form of "site:domain") that can be submitted to popular search engines to restrict the outcome to a particular domain. In the result page, both Google and Yahoo (now part of Bing) offer an estimated count of how many pages from that domain are contained in their index. We have verified that these counts are exact for small domains and have no reason to doubt their ballpark accuracy for larger domains. Running site queries for all gTLDs and cc-TLDs allows one to obtain not just the index size of a search engine, but also its distribution of pages between the various top-level domains. The results are summarized in Table III, which shows that Google's self-reported index at that time contained 30.7B pages, while Yahoo's 37.8B.

In order to compare the coverage within each TLD, we designate one of the sets as the *base* and sort all domains in the descending order of the number of pages in the base dataset. Furthermore, instead of using raw page counts, which are functions of crawl size, we are more interested in *fractions* of each crawl allocated to each TLD. To highlight this better, Table IV shows the top-10 list using Google as the base (for the three academic crawls, we use only 200-OK HTML pages).

We include a WebBase crawl that took place concurrently with IRLbot's and ClueWeb09 (both already detailed in Table II) to highlight the fact that individual crawl policy may purposely favor skewed allocation of resources across domains, leading to a drastically different Internet coverage from that of other crawlers. In this case, the difference occurs because WebBase was interested only in specific sites, while ClueWeb09 targeted static pages in 10 pre-selected languages.

Fig. 7 plots the entire TLD curve using Google as the base. Yahoo's deviation at the beginning of the plot in part (a) is noticeably smaller than that of IRLbot in part (b); however, eventually the two curves exhibit random oscillations of similar magnitude. A closer look at the 40 most-popular domains in subfigures (c)-(d) reveals that IRLbot's biggest discrepancy appears in three points – `.edu` (#12), `.gov` (#24), and `.info` (#15) – where the first two are under-crawled and the third one is over-crawled.

The former case can be explained by IRLbot's budget function that favored TLDs with many individual domains. Both `.edu` and `.gov` contained a small number of unique PLDs, which despite their high ranking were given a relatively low aggregate budget in comparison to all other domains. The issue with `.info` can be traced to the large number of $0.99/year spam PLDs hosted there at the time, which conceivably were either removed from Google's index or significantly throttled down during crawling using techniques that were not available to IRLbot (e.g., content analysis, ranking from prior crawls, user click behavior).

In general, performing analysis of TLD coverage helps one detect over/under-represented parts of the web in crawl data, identify spam regions, and tune budget-allocation policies, all of which are beneficial tools for future crawler development.

VII. Conclusion

This paper presented new IRLbot implementation details, proposed a novel methodology for documenting large-scale crawls, and used it to deliver a massive amount of previously undocumented information about the IRLbot experiment. We also derived a model for extrapolating the growth rate of unique nodes (e.g., pages, hosts, and PLDs) as a function of crawl size, confirming the colloquial notion that the space of URLs and hostnames is infinite, and estimated the number of remaining PLDs in larger IRLbot crawls. We finally proposed several methods for assessing Internet-wide crawl coverage, examined the budget function of IRLbot, and suggested avenues for improvement.

References

[1] S. Abiteboul, M. Preda, and G. Cobena, "Adaptive on-line page importance computation," in *Proc. WWW*, May 2003, pp. 280–290.

[2] S. T. Ahmed and D. Loguinov, "Modeling Randomized Data Streams in Caching, Data Processing and Crawling Applications," in *Proc. IEEE INFOCOM*, Apr. 2015.

[3] M. H. Alam, J. Ha, and S. Lee, "Novel Approaches to Crawling Important Pages Early," *Springer Knowledge and Information Systems*, Sep. 2012.

[4] J. Alpert and N. Hajaj, "We Knew the Web Was Big..." Jul. 2008. [Online]. Available: http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html.

[5] R. Baeza-yates and C. Castillo, "Crawling the infinite Web: five levels are enough," in *Proc. WAW*, 2004, pp. 156–167.

[6] R. Baeza-Yates, C. Castillo, M. Marin, and A. Rodriguez, "Crawling a Country: Better Strategies than Breadth-First for Web Page Ordering," in *Proc. WWW*, May 2005.

[7] K. Bharat, B.-W. Chang, M. Henzinger, and M. Ruhl, "Who Links to Whom: Mining Linkage between Web Sites," in *Proc. IEEE ICDM*, Nov. 2001.

[8] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "UbiCrawler: A Scalable Fully Distributed Web Crawler," *Software: Practice & Experience*, vol. 34, no. 8, pp. 711–726, Jul. 2004.

[9] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," in *Proc. WWW*, Apr. 1998, pp. 107–117.

[10] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener, "Graph Structure in the Web," *Computer Networks*, vol. 33, pp. 309–320, Jun. 2000.

[11] A. Z. Broder, M. Najork, and J. L. Wiener, "Efficient URL Caching for World Wide Web Crawling," in *Proc. WWW*, May 2003, pp. 679–689.

[12] M. Burner, "Crawling Towards Eternity: Building an Archive of the World Wide Web," *Web Techniques Magazine*, vol. 2, no. 5, May 1997.

[13] J. Callan, M. Hoy, C. Yoo, and L. Zhao, "The ClueWeb09 Dataset," Nov. 2009. [Online]. Available: http://boston.lti.cs.cmu.edu/classes/11-742/S10-TREC/TREC-Nov19-09.pdf.

[14] C. Castillo, M. Marin, A. Rodriguez, and R. Baeza-Yates, "Scheduling Algorithms for Web Crawling," in *Proc. Latin American Web Conference*, Oct. 2004.

[15] A. Chandramouli, S. Gauch, and J. Eno, "A Popularity-Based URL Ordering Algorithm for Crawlers," in *Proc. IEEE HSI*, May. 2010, pp. 259–263.

[16] J. Cho and H. Garcia-Molina, "Parallel Crawlers," in *Proc. WWW*, May 2002, pp. 124–135.

[17] J. Cho, H. Garcia-Molina, T. Haveliwala, W. Lam, A. Paepcke, and S. R. G. Wesley, "Stanford WebBase Components and Applications," *ACM Trans. Internet Technology*, vol. 6, no. 2, pp. 153–186, May 2006.

[18] J. Cho, H. Garcia-Molina, and L. Page, "Efficient Crawling through URL Ordering," in *Proc. WWW*, Apr. 1998, pp. 161–172.

[19] J. Cho and U. Schonfeld, "RankMass Crawler: A Crawler with High PageRank Coverage Guarantee," in *Proc. VLDB*, Sep. 2007.

[20] ClueWeb09 Dataset. [Online]. Available: http://www.lemurproject.org/clueweb09/.

[21] N. Eiron, K. S. McCurley, and J. A. Tomlin, "Ranking the Web Frontier," in *Proc. WWW*, May 2004, pp. 309–318.

[22] M. A. Golshani, V. Derhami, and A. ZarehBidoki, "A Novel Crawling Algorithm for Web Pages," in *Proc. AIRS*, 2011, pp. 263–272.

[23] A. Heydon and M. Najork, "Mercator: A Scalable, Extensible Web Crawler," *World Wide Web*, vol. 2, no. 4, pp. 219–229, Dec. 1999.

[24] Y. Hirate, S. Kato, and H. Yamana, "Web Structure in 2005," in *Proc. WAW*, Nov. 2006, pp. 36–46.

[25] L. Huang, J. J. H. Zhu, and X. Li, "Histrace: Building a Search Engine of Historical Events," in *Proc. WWW Poster Session*, Apr. 2008, pp. 1155–1156.

[26] Internet Archive. [Online]. Available: http://archive.org/.

[27] Q. Jiang and Y. Zhang, "SiteRank-Based Crawling Ordering Strategy for Search Engines," in *Proc. CIT*, Oct. 2007, pp. 259–263.

[28] H.-T. Lee, D. Leonard, X. Wang, and D. Loguinov, "IRLbot: Scaling to 6 Billion Pages and Beyond," in *Proc. WWW*, Apr. 2008, pp. 427–436.

[29] B. T. Loo, S. Krishnamurthy, and O. Cooper, "Distributed Web Crawling over DHTs," EECS Dept., University of California, Berkeley, Tech. Rep. UCB/CSD-04-1305, 2004.

[30] O. A. McBryan, "GENVL and WWWW: Tools for Taming the Web," in *Proc. WWW*, May 1994.

[31] K. Moody and M. Palomino, "SharpSpider: Spidering the Web through Web Services," in *Proc. IEEE LA-WEB*, Nov. 2003, pp. 219–221.

[32] M. Najork and A. Heydon, "High-Performance Web Crawling," Compaq SRC, Tech. Rep. 173, Sep. 2001. [Online]. Available: http://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-173.pdf.

[33] M. Najork and J. L. Wiener, "Breadth-First Search Crawling Yields High-Quality Pages," in *Proc. WWW*, May 2001, pp. 114–118.

[34] NSF, "A CluE in the Search for Data-Intensive Computing." [Online]. Available: http://www.nsf.gov/news/news_summ.jsp?cntn_id=111470.

[35] Nutch. [Online]. Available: http://nutch.apache.org/.

[36] B. Pinkerton, "Finding What People Want: Experiences with the Web Crawler," in *Proc. WWW*, Oct. 1994.

[37] V. Shkapenyuk and T. Suel, "Design and Implementation of a High-Performance Distributed Web Crawler," in *Proc. IEEE ICDE*, Mar. 2002, pp. 357–368.

[38] A. Signhal, "Breakfast with Google's Search Team," Aug. 2012. [Online]. Available: https://www.youtube.com/watch?v=8a2VmxqFg8A.

[39] A. Singh, M. Srivatsa, L. Liu, and T. Miller, "Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web," in *Proc. SIGIR Workshop on Distributed Information Retrieval*, Aug. 2003, pp. 126–142.

[40] T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasundaram, "ODISSEA: A Peer-to-Peer Architecture for Scalable Web Search and Information Retrieval," in *Proc. WebDB*, Jun. 2003, pp. 67–72.

[41] H. Yan, J. Wang, X. Li, and L. Guo, "Architectural Design and Evaluation of an Efficient Web-crawling System," *J. Systems and Software*, vol. 60, no. 3, pp. 185–193, Feb. 2002.

[42] J. J. H. Zhu, T. Meng, Z. Xie, G. Li, and X. Li, "A Teapot Graph and Its Hierarchical Structure of the Chinese Web," in *Proc. WWW Poster Session*, Apr. 2008, pp. 1133–1134.