# Documenting a large-scale IRLBot crawl

# Paper Goals

- Issues today:
  - Assumption: efficient crawling needs heavily parallelized architecture, more hardware, thus great financial investment
  - Prior crawls are usually not documented too well
    - No standard way to compare crawling performances
    - Lack of transparency of crawling in industry giants
  - No provisions to handle spam

- Overview:
  - New method proposed to analyze web crawl performance
  - Break down IRLBot crawl experiment
  - Compare crawl coverage to commercial search engines

# Understanding web crawls

Crawling process, challenges, prior work

- Crawler operation
  - Crawl cycle:
    - Parsing (maintaining concurrent HTTP sessions and parsing HTML)
    - Eliminate duplicate URLs
    - Frontier ranking (seen but not-yet-crawled URLs ranking)
    - Admission control on pending URLs using ranks computed in real-time
    - DNS lookup
    - Enforce robots.txt directives of crawled websites
    - Adhere to politeness rate limits that prevent crashing of individual servers
  - Factors:
    - D: number of downloaded pages
    - q: fraction of D that is error-free HTML content
    - h: number of crawl servers
    - l: average locally unique links per page
    - p: fraction of l that is globally unique
    - S: target crawl speed
    - Crawler design is a tradeoff between {D/h, S/h, q, l}

- Duplicate elimination
  - First generation:
    - Either kept all data in RAM or used random disk access to verify URL uniqueness
  - Second generation:
    - Disk-seek replaced with batch-sort that periodically scanned file of previously-seen URLs and merged new URLs in
  - Third generation:
    - Focused on horizontal scalability (increasing h)
    - Parallelize URL workload across server clusters/p2p networks

- Ranking and admission control [comparison with other implementations]
  - Previous literature doesn't use real-time spam avoidance or global frontier prioritization
  - Other approaches such as OPIC and PageRank:
    - Compute graph-theoretic metrics for ranks
    - Use offline calculations due to high input/output and CPU cost
  - Open-source implementations don’t publish performance/operational details and need substantial resources (high h)

- Discussion
  - Aim should be to surpass prior crawls in all four parameters, i.e. {D/h, S/h, q, l}
  - Important factors for analysis: scalability and average crawl depth
  - Why consider average crawl depth?
    - Controls spam likelihood, number of crawled hosts/domains/IPs (cache size), DNS/robots.txt workload, complexity of politeness rate-limiting, **internet coverage**
  - IRLBot specifications:
    - m(num of seed nodes)=h=1, max q, unrestricted l, S/D outside control (based on university bandwidth)

# Page-level Analysis

Proposed method for documenting (large-scale) IRLBot crawl

- Admitted URLs
  - IRLBot handling redirects in normal URLs:
    - Avoid spending bandwidth on lengthy redirect spam
    - Each 301 and 302 redirect HTTP is treated as new link- sent it for regular uniqueness verification and then admission control
    - Redirects need to pass spam-related budget enforcement before reattempt
    - Makes retry latency dependent on corresponding domain's current rank/URL backlog
- Crawled URLs
  - Possible reasons for failure:
    - Connect and receive failures, spammer stalling tactics: host did not provide data in time/dragged out download for long, serving infinite data streams, missing status line in HTTP response, failed decompression: gzip corruption, bogus encoding, invalid HTTP status code, unparsable URL, violated chunking syntax/exceeded max size on unchunking, contained HTTP headers over max size
  - Focus on crawling HTML pages

- Downloaded URLs
  - The "accept: text/html" header with all non-robots.txt requests is universally ignored by internet servers

- Links
  - Ignored <img>
  - Tested links for correctness of syntax (invalid syntax/excessive length)
  - Extensive black-list of non-HTML extensions: did not reduce workload enough, filter can be discarded
  - Removed same-page duplicates
  - Web-graph created by replacing URLs with 64-bit hashes to feed into admission control in URL cycle

# Server-level Analysis

Documenting network interactions

- DNS and robots
  - Crawler interaction with remote hosts and their authoritative DNS servers
  - IRLBot only issued DNS queries for URLs passing budget enforcer
  - If a website fails to provide legit robots.txt, it prevents IRLBot from knowing which parts of website to exclude, and so the whole host treated as non-crawlable
  - Servers sending HTTP fillers (custom error messages, redirects to default pages, ads) instead of proper errors: provided robot files with no content-type/non-text/plain type, assumed equivalent to not having any crawling restrictions
  - Retaining only the directives that applied to either all crawlers or IRLBot specifically

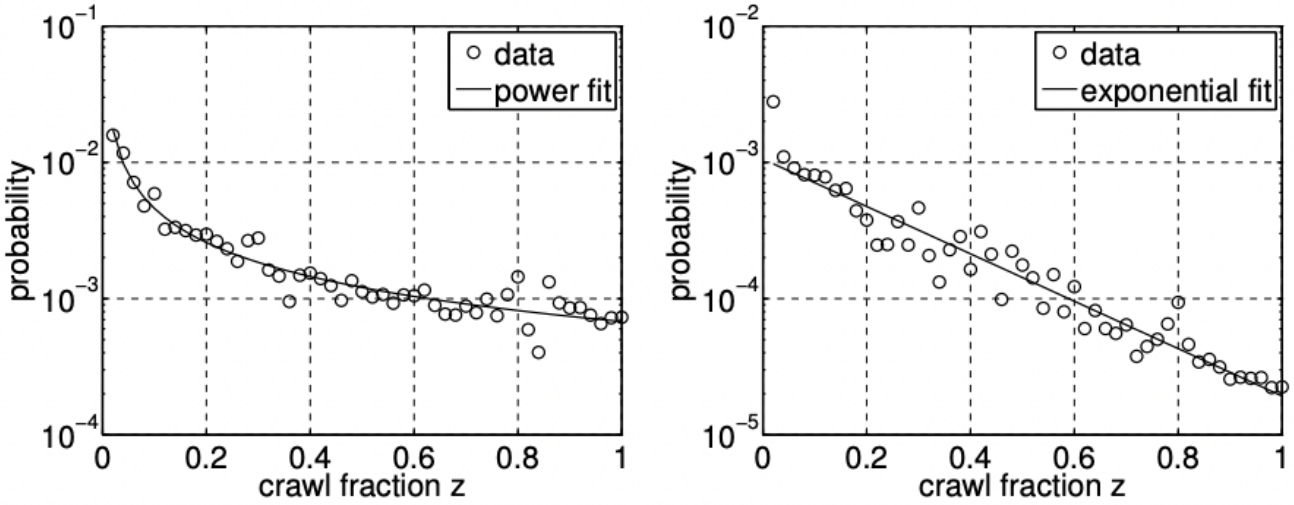
# Extrapolating Crawls

Crawl dataset growth and finiteness discussion

- Stochastic model
  - Webgraph of internet represented as G = (V, E)
  - Crawl viewed as stochastic process {(Xn, Yn)}: n = time, Xn = crawled page that generated link n, Yn = URL it points to
  - Indicator var Qn = 1 if link (Xn, Yn) satisfies some uniqueness condition (eg. Yn not seen before) and 0 otherwise
  - Expected number of links LN satisfying Qn in crawl of size N
  - $$E[L_N] = \sum_{n=1}^{N} E[Q_n] \approx \int_1^N p(t)dt$$
  - p(t) is the growth rate of unique nodes at time t
  - Expectation: p(t) starts high for small t, eventually Lt should start approaching saturation and p(t) should become 0

- Data extraction
  - MapReduce algorithm to estimate p(t)
  - Define bins [ti-Δ, ti+Δ]
  - Map-> for each link (j, k) found in page j:
    - Find bin based on j's crawl timestamp (τj)
    - Increment seen out-links (si) for it
    - Map (j, k) to k's hash (hk) and τj [i.e. <hk, τj>]
    - Sort URLs by hk
  - Reduce-> retain smallest timestamp for each seen URL
  - For each bin p(ti) = ui (globally unique links in bin i)/si

- URLs
  - K = already-crawled portion of the web
  - z = t/K = time normalized to this crawl
  - p'(z) = p(zK) = corresponding uniqueness function
  - 
  - r = N (nu $E[L_N] \approx \mathcal{K} \int_0^r \tilde{p}(z)dz$ to generate LN globally unique nodes)/K
  - Predicted: infinite hostnames and URLs




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

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

# Internet-wide Coverage

Crawl coverage discussion

- Basic properties
  - Crawl coverage: graph of visible web includes
    - URLs returning 200 OK HTML content
    - Nodes in frontier
    - HTTP errors (provide info about redirects, dead nodes, forbidden URLs, parents of crawled pages -> useful for merging duplicate pages, spam detection, general page tracking, backtracking crawl tree for complaints)
    - Links connecting them together

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 | ∼ 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 |

- TLD coverage
  - Allocation of budgets to individual domains
  - Understanding how much of crawler bandwidth is spent in what parts of the Internet
  - Site queries (site:domain) to restrict outcome to a single domain -> how many pages of a domain in the total index
  - Compare TLD coverage:
    - One set is base; sort domains in descending order of page count in base dataset
    - Use fractions of crawl allocated to each TLD
    - IRLBot favored TLDs with many individual domains
  - TLD coverage analysis helps detect over/underrepresented parts of web in crawl data, understanding how much of crawler bandwidth is spent in what parts of the Internet

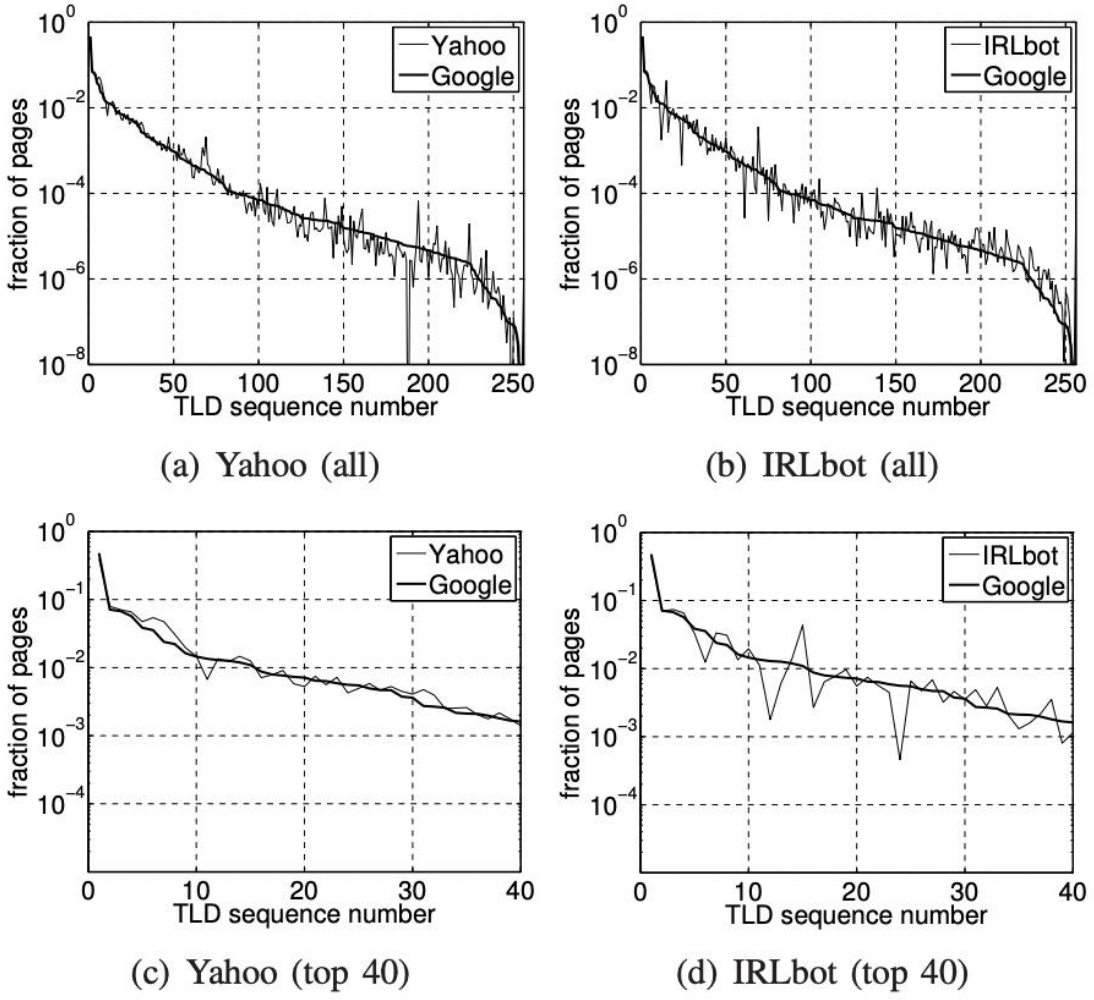



Fig. 7. TLD coverage (Google order).

# Reference

S. T. Ahmed, C. Sparkman, H. -T. Lee and D. Loguinov, "**Around the web in six weeks: Documenting a large-scale crawl**," 2015 IEEE Conference on Computer Communications (INFOCOM), Hong Kong, China, 2015, pp. 1598-1606, doi: 10.1109/INFOCOM.2015.7218539.