Optimal Freshness Crawl Under Politeness Constraints

Andrey Kolobov
Microsoft Research
Redmond, WA, USA
akolobov@microsoft.com

Yuval Peres
yperes@gmail.com

Eyal Lubetzky
Courant Institute of Mathematical Sciences
New York University
New York, NY, USA
eyal@courant.nyu.edu

Eric Horvitz
Microsoft Research
Redmond, WA, USA
horvitz@microsoft.com

ABSTRACT

A Web crawler is an essential part of a search engine that procures information subsequently served by the search engine to its users. As the Web is becoming increasingly more dynamic, in addition to discovering new web pages a crawler needs to keep revisiting those already in the search engine's index, in order to keep the index *fresh* by picking up the pages' changed content. Determining how often to recrawl pages requires making tradeoffs based on the pages' relative importance and change rates, subject to multiple resource constraints — the limited daily budget of crawl requests on the search engine's end and politeness constraints restricting the rate at which pages can be requested from a given host. In this paper, we introduce PoliteBinaryLambdaCrawl, the first optimal algorithm for freshness crawl scheduling in the presence of politeness constraints as well as non-uniform page importance scores and the crawler's own crawl request limit. We also propose an approximation for it, stating its theoretical optimality conditions and in the process discovering a connection to an approach previously thought of as a mere heuristic for freshness crawl scheduling. We explore the relative performance of PoliteBinaryLambdaCrawl and other methods for handling politeness constraints on a dataset collected by crawling over 18.5M URLs daily over 14 weeks.

CCS CONCEPTS

• **Information systems → Web search engines**; **Web crawling**; *Information retrieval*; • **Computing methodologies → Planning under uncertainty**;

KEYWORDS

Web crawling; search engine; politeness constraint; convex optimization; Lagrange multiplier; planning under uncertainty

ACM Reference Format:
Andrey Kolobov, Yuval Peres, Eyal Lubetzky, and Eric Horvitz. 2019. Optimal Freshness Crawl Under Politeness Constraints. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '19), July 21–25, 2019, Paris, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3331184.3331241

SIGIR '19, July 21–25, 2019, Paris, France

ACM ISBN 978-1-4503-6172-9/19/07…$15.00
https://doi.org/10.1145/3331184.3331241

1 INTRODUCTION

A Web crawler is an essential part of a search engine that procures information subsequently served by the search engine to its users. It does so by downloading the content of URLs that the search engine is aware of, and storing their processed versions into the search engine's index, from which information relevant to a query can be retrieved efficiently. The content of some of the URLs the crawler visits is entirely new to the index. However, most of the web pages the crawler downloads on any given day were by it multiple times in the past. The crawler revisits them in order to keep the search engine's index — and its search results — *fresh*. Indeed, serving stale content such as YouTube pages with taken-down videos or failing to serve pages that the user *knows* to be relevant to her query due to new content is a cause of user dissatisfaction.

Scheduling a freshness crawl, i.e., determining which web pages in the index to re-download and when, must take into account multiple resource constraints. On its own end, the number of crawl requests a crawler can issue daily is constrained by the physical limitations of the search engine's infrastructure. On the other end, the crawler must respect *politeness constraints* imposed by the hosts it wishes to download the content from. Much like the crawler, hosts' infrastructure and financial feasibility limit the number of requests they can serve in a given time period, and hosts have preferences on how to allocate this budget. Many are willing to allocate only a fraction of it to a search engine, and only as long as the search engine brings them customers. These constraints further complicate the problem of determining the optimal recrawl policy: this optimization problem must take into account both the importance of every web page and its change rate, and needs to be have an algorithm efficient enough to handle very large page sets.

Our work introduces PoliteBinaryLambdaCrawl, the first optimal algorithm for freshness crawl scheduling in the presence of non-uniform page importance scores and politeness as well as the crawler's own request rate constraints. Freshness crawl scheduling without regard to politeness has been studied extensively [2, 6–10, 16, 22] starting with a work by Coffman et al. [11]. We focus on the mathematical notion of freshness due to Cho and Garcia-Molina [8] and use an optimization algorithm for it due to Azar et al. [2] as PoliteBinaryLambdaCrawl's subroutine. At

a high level, PoliteBinaryLambdaCrawl iteratively "fixes" the politeness-unconstrained policy computed by Azar et al. [2]. Our key contribution is proving that this process is optimal and showing its theoretical and empirical efficiency. Researchers studied freshness maintenance under politeness constraints of a *single* information source [12], such as a host, and applied this result to cache synchronization [5, 20]. However, to our knowledge, no prior work has researched freshness optimization for many URLs under many politeness constraints. So far, the main approach to ensuring politeness has been throttling back the request rate to match the host's expectation at the level of crawler architecture [3, 21].

Specifically, this work's contributions are:

- We present and prove the optimality of PoliteBinaryLambdaCrawl, a new scalable algorithm for computing a freshness crawl policy under multiple host politeness constraints and the crawler's own crawl rate constraint, in the presence of non-uniform URL importance scores and change rates.
- Using PoliteBinaryLambdaCrawl's proof of optimality, we derive an approximation method that does not depend on page change rates for crawl scheduling. Its politeness and optimality depend on a condition on the distribution of URLs' importance scores and change rates that we identify. This result simultaneously provides the first optimality guarantee for importance-proportional crawl scheduling [8], previously thought to be a mere heuristic.
- We evaluate PoliteBinaryLambdaCrawl on a dataset of 18.5 million URLs crawled daily for 14 weeks. We investigate the performance of PoliteBinaryLambdaCrawl and strong alternatives, including the importance-based approximation mentioned above. We assess the approximation's sensitivity to the distribution of URLs' importance and change characteristics. Finally, we show that in practice PoliteBinaryLambdaCrawl scales even better than theory predicts.

2 BACKGROUND AND ASSUMPTIONS

Before presenting our own algorithm, we briefly survey the relevant terms and describe a previously proposed crawl scheduling algorithm that serves as a building block for ours.

2.1 Web Crawl

Web crawl is the process of downloading web pages' content from the corresponding URLs, e.g., by a *search engine*. The search engine records this content in an *index* and uses it to determine the URLs' relevance to incoming queries. Processed web page data may also be served via question answering services or digital assistants. Thus, the *freshness* of web page content in a search engine's index, which for now we define loosely as a measure of discrepancy between a URL's actual content and its indexed copy, is a major factor in the quality of a search engine and services that rely on it.

The problem of *freshness crawl scheduling* consists in determining a *crawl schedule* (i.e., download schedule) for a set of web pages W, which we assume to be fixed, so as to maximize the freshness of their copies in the search engine's index. Freshness crawl is related to but generally different from *coverage crawl*, whose objective is to discover new URLs on the Web [18]. There are several types of crawl schedules. In this paper, we assume that each URL's is crawled according to a time-homogeneous Poisson process:

Definition 2.1. A *time-homogeneous randomized crawl schedule* (called *crawl schedule* in the rest of the paper) for a set W of n web pages indexed by $\{1, \ldots, n\}$ is a vector of values $(\rho_1, \ldots, \rho_n)$, called *crawl rates*, where ρ_i is the Poisson rate for page i's crawl.

2.2 Constraints on Crawl Rate

Web pages are located on machines called *hosts* that have associated IP-addresses — possibly more than one. Generally, web masters do not want search engines to request pages from hosts arbitrarily often, since this reduces resources for serving other clients, and impose *politeness constraints* on crawl rates. Like the notion of crawl schedules, politeness constraints can be defined in several ways. We use the following definition for host-associated politeness constraints, with IP-associated constraints defined analogously:

Definition 2.2. For given host h with the set W_h of all pages on it, a *politeness expectation constraint* (called *politeness constraint* in the rest of the paper) is a server-imposed crawl rate limit R_h such that

$$\sum_{i \in W_h} \rho_i \leq R_h \,.$$

In reality the intent of a politeness constraint is to limit the *maximum*, not *expected* number of page requests that can be issued by a crawler to a host in a given unit of time, e.g., per day. However, in practice short-term crawl rate spikes violating politeness constraints usually do not trigger punitive measures from the host if the crawl rate is below the constraint on average across several days. Thus, the above expectation-based definition is justified.

Theoretically, a web page can be subject to both a per-host and a per-IP politeness constraint. In practice, however, it is overwhelmingly host constraints that limit crawl rates. If a host has several IP addresses with a constraint on each, in addition to a proper per-host constraint, then the sum of per-IP constraints is almost always more lax the per-host constraint. In other words, IP constraints mainly force the crawler to choose the address at which a page download can be requested but don't restrict downloads from a host per se. Moreover, hosts that have only associated per-IP constraints tend to have pages that change very infrequently, aren't very popular, or both, so per-IP constraints don't limit search engines' crawls in this case either. Per-IP politeness constraints' purpose is usually different than limiting host access rates; therefore, in this paper *we focus on crawl scheduling under per-host politeness constraints only.*

Last but not least, a search engine's web crawl is also limited by the search engine's own capacity:

Definition 2.3. A *request rate constraint* is a search engine-imposed crawl rate limit R such that

$$\sum_{i=1}^{n} \rho_i \leq R$$

for the set of all web pages $\{1, \ldots, n\}$ in the search engine's index.

2.3 Freshness crawl optimization

Freshness crawl optimization has been studied under a variety of assumptions. We review that research in the Related Work section, and here focus on the model [2] that serves as our starting point.

That model associates an *importance* score μ_i and a Poisson *change rate* Δ_i with every page $1 \leq i \leq n$. The model implicitly

assumes that each page changes in a time-homogeneous way. While not entirely accurate — a page's change times can have notable patterns [22] — this model has the advantages of being reasonably close to reality [4, 6] and mathematically convenient. Importance μ_i can also be thought of as the time-homogeneous Poisson rate at which the page is served in response to the query stream, but can in general be any positive stationary measure of a page's significance. As with the change rate, despite spikes in pages' popularity/importance, this assumption is a good tradeoff between accuracy and computational cost. At request time t, a page i's freshness is binary-valued, given by an indicator variable $\mathbb{1}_{\text{Fresh}(i,t)}$ having value 1 if and only if page i hasn't changed by time t since it was last recrawled.

Under this model, the goal is to optimize the *time-averaged expected freshness* across all pages over a large time horizon T, i.e.

$$F_T^* = \lim_{T\to\infty} \mathbb{E}\left[\frac{1}{T}\int_0^T \left(\sum_{i=1}^n \mu_i \cdot \mathbb{1}_{\text{Fresh}(i,t)}\right) dt\right] \tag{1}$$

Azar et al. [2] demonstrate that for large page sets and under the assumption of page changes being governed by Poisson processes, finding the optimal randomized time-homogeneous scheduling policy — a vector of values $\vec{\rho} = (\rho_1, \ldots, \rho_n)$ such that ρ_i is the Poisson rate for page i – maximizing this expectation under the request rate constraint amounts to solving the following optimization problem:

Problem 1 (Continuous-time policy under crawler's page request rate constraint).

Input: Request rate constraint $R > 0$ and importances and change rates $\vec{\mu}, \vec{\Delta} = \{\mu_i, \Delta_i\}_{i=1}^n$ s.t. $\mu_i, \Delta_i > 0$ for all pages i in W.

Output: Update rates $\vec{\rho}^* = (\rho_1^*, \ldots, \rho_n^*)$ satisfying $\rho_i \geq 0$ for all i and $\sum_{i=1}^n \rho_i = R$, and maximizing

$$F(\vec{\rho}) = \sum_{i=1}^n \frac{\mu_i \rho_i}{\rho_i + \Delta_i}. \tag{2}$$

$F(\vec{\rho})$ denotes the *policy value* for the policy parameterized by $\vec{\rho}$.

If it wasn't for the $\rho_i \geq 0$ conditions for all i, which we call *semantic constraints*, this would be an ordinary optimization problem under an equality constraint (crawler's request rate constraint $g(\vec{\rho}) = \sum_i \rho_i = R$). Conceptually, the proposed optimization procedure [2], summarized in Algorithm 1, consists of two stages:

(1) Ignore the $\rho_i \geq 0$ conditions and solve Problem 1 as optimization under an equality constraint, using Lagrange multipliers. This method yields the following system of equations:

$$\nabla F(\vec{\rho}) = \lambda \nabla g(\vec{\rho})$$
$$\sum_{i=1}^n \rho_i = R$$

Taking partial derivatives of F and g and other simple algebraic manipulations yield λ, the Lagrange multiplier, and a vector of quasi-rates $\vec{\rho} = (\rho_1, \ldots, \rho_n)$, some of which may be negative. The solution is

$$\rho_i = \sqrt{\frac{\mu_i \Delta_i}{\lambda}} - \Delta_i \qquad (i = 1, \ldots, n) \tag{3}$$

$$\lambda = \left(\frac{\sum_i \sqrt{\mu_i \Delta_i}}{R + \sum_i \Delta_i}\right)^2 \tag{4}$$

(2) Convert $\vec{\rho}$ to a vector $\vec{\rho}^*$ of actual optimal recrawl rates using an algorithm that sorts pages in the non-decreasing order of $\frac{\mu_i}{\Delta_i}$ and finds i^* s.t. Equations (3) and (4), when solved *only* for pages with $i \geq i^*$ in this ordering, yield only $\rho_i > 0$ for all $i \geq i^*$. Lines 3-9 of Algorithm 1 describe the search for this threshold value precisely. These ρ_i values are the optimal recrawl rates for pages $i \geq i^*$. For pages $i < i^*$, $\rho_i^* = 0$.

Algorithm 1: BinaryLambdaCrawl: Optimal Continuous-time Policy Under Crawler's Request Rate Constraint

Input: W – a set of web pages; $R > 0$ – request rate constraint; $\vec{\mu}, \vec{\Delta}$ – importance and change rate vectors of length n.
Output: $\vec{\rho}$ – the vector specifying optimal the crawl rate for each page $i \in W$

1 Sort μ_i, Δ_i in non-descending order of μ_i/Δ_i for pages $i \in W$
2 Let

$$r \leftarrow \sum_{i=1}^n \sqrt{\mu_i \Delta_i} \quad , \quad s \leftarrow \sum_{i=1}^n \Delta_i \, .$$

3 **for** $i = 1$ **to** n **do**
4 **if** $\mu_i/\Delta_i \leq \left(\frac{r}{R+s}\right)^2$ **then**
5 $\rho_i \leftarrow 0$
6 $r \leftarrow r - \sqrt{\mu_i \Delta_i}$
7 $s \leftarrow s - \Delta_i$
8 **else**
9 $\rho_i \leftarrow \sqrt{\mu_i \Delta_i}(R + s)/r - \Delta_i$
10 Return $\vec{\rho}$

The correctness of Algorithm 1 is established by the following claims [2]:

Theorem 2.4. *Function F (Equation (2)) is strictly concave on $[0, R]^n$ and has a unique optimal solution $\vec{\rho}^*$.*

Theorem 2.5. *Assuming that web pages are sorted s.t. $\frac{\mu_1}{\Delta_1} \leq \ldots \leq \frac{\mu_n}{\Delta_n}$, Step 2 of the above procedure yields the optimal $\vec{\rho}^*$.*

Although Azar et al. [2] do not give this algorithm a name, we refer to it as BinaryLambdaCrawl throughout this paper.

3 OPTIMAL CRAWL UNDER POLITENESS CONSTRAINTS

To describe our approach to finding optimal crawl policy under politeness as well as request rate constraints, we first formalize this problem, then present our high-level algorithmic strategy and intuition behind it, describe the algorithm formally, and finally give a proof of its correctness.

3.1 The Formal Model

Before presenting the model and the algorithm, we summarize the notation. Let

- W be the set of pages of size n for which we are computing a recrawl policy.
- $\bigcup_{h=1}^{m} S_h = W$ be a partition of page set W into m subsets, with each $1 \le h \le m$ corresponding to a host and $S_h \subseteq W$ being the set of pages belonging to host h. A page can belong to only one host, hence for any two distinct h and h', $S_h \bigcap S_{h'} = \emptyset$.
- R_h be host h's politeness constraint, s.t. $\sum_{i \in W_h} \rho_i \le R_h$.
- R be the request rate constraint, i.e., $\sum_{i \in W} \rho_i \le R$.

In practice, many pages belong to hosts without associated politeness constraints. Theoretically crawlers are allowed to download pages from such hosts virtually infinitely often, but realistically pages from these hosts just aren't popular or fast-changing enough to warrant frequent recrawls. Mathematically, we don't need to distinguish between different such hosts, and therefore assume w.l.o.g. that there is exactly one host h for which $R_h = \infty$. We also assume that each finite politeness constraint R_h is smaller than our request rate constraint: $R_h < R$. Finally, since we have a host with an infinite politeness constraint, letting $\overline{R} = \sum_h R_h$ we have $R \le \overline{R}$.

With these definitions in mind, finding the optimal page recrawl rates amounts to solving the optimization problem below:

Problem 2 (Randomized continuous-time policy under politeness and request rate constraints).

Input: crawler request rate constraint $R > 0$; importance scores and change rates $\mu_i, \Delta_i > 0$ for $i = 1, \ldots, n$; host partition $\bigcup_{h=1}^{m} S_h = W = \{1, \ldots, n\}$, host politeness constraints $0 < R_h \le R$ for each h whenever $R_h < \infty$ such that $\overline{R} := \sum_h R_h \ge R$;

Output: Recrawl rates $\vec{\rho}^* = (\rho_1, \ldots, \rho_n)$

$$\text{maximizing} \quad F(\vec{\rho}) = \sum_{i \in W} \frac{\mu_i \rho_i}{\rho_i + \Delta_i}$$

$$\text{subject to} \quad \sum_{i \in W} \rho_i = R \qquad \text{(request rate constraint)}$$

$$\sum_{i \in S_h} \rho_i \le R_h \quad (\forall 1 \le h \le m) \qquad \text{(politeness constraints)}$$

$$\vec{\rho} \ge 0 \qquad \text{(semantic constraints)}$$

3.2 The Approach

Solving Problem 2 requires non-linear optimization under *inequality* constraints, which, in general, could take time exponential in the constraint number. Setting the semantic constraints aside for the moment, the difficulty is in identifying the subset of the politeness constraints that are *saturated* under the optimal solution $\vec{\rho}^*$:

Definition 3.1. A recrawl rate vector $\vec{\rho}$ *saturates* a constraint R_h if $\sum_{i \in S_h} \rho_i = R_h$.

If we knew which politeness constraints Problem 2's optimal solution saturates, we could treat them as equalities during optimization, solving for them with the method of Lagrange multipliers.

Our main result is that one can find the optimal solution to Problem 2 efficiently by considering only $O(m)$ constraint subsets, using the following high-level approach:

(1) Solve the optimization as in Problem 1 under the request rate constraint R only, *ignoring all politeness constraints*, using BinaryLambdaCrawl. Since we assume $\overline{R} := \sum_h R_h \ge R$, the optimal solution $\vec{\rho}^*$ for Problem 2 saturates the crawler's request rate constraint *even if politeness constraints are taken into account*. Determine the hosts whose politeness constraints the optimal solution $\vec{\rho}_0^*$ of this step violates.

(2) For each host constraint R_h violated by $\vec{\rho}_0^*$, apply BinaryLambdaCrawl to the pages on that host, treating R_h as an equality constraint. Then remove this host's page set S_h from further consideration, and subtract crawl rate R_h from the overall crawl rate constraint R. Go to step (1) and keep iterating until step (1)'s solution (on the remaining pages and crawl rate budget!) obeys all politeness constraints.

Algorithm 2: PoliteBinaryLambdaCrawl: optimal algorithm for Problem 2

Input: $R > 0$ – crawler's request rate constraint;
$\vec{\mu}, \vec{\Delta}$ – importance and change rate vectors of length n each;
host partition $W = \bigcup S_h$ and host politeness constraints $0 < R_h \le R$

Output: $\vec{\rho}^*$ – vector of length n specifying the crawl rate per page.

1 $\vec{\rho}'^* \leftarrow$ BinaryLambdaCrawl$(R, \vec{\mu}, \vec{\Delta})$ // ignore politeness
2 **if** $\sum_{i \in S_h} \rho_i'^* \le R_h$ *for every* h **then** // politeness not violated
3 $\quad \vec{\rho}^* \leftarrow \vec{\rho}'^*$
4 $\quad$ Return $\vec{\rho}^*$
5 **foreach** h *s.t.* $\sum_{i \in S_h} \rho_i'^* > R_h$ **do** // saturate violated constraint
6 $\quad \vec{\rho}'^* \restriction_{S_h} \leftarrow$ BinaryLambdaCrawl$(R_h, \vec{\mu} \restriction_{S_h}, \vec{\Delta} \restriction_{S_h})$. // restrict to S_h
7 $\quad \vec{\rho}^* \restriction_{S_h} \leftarrow \vec{\rho}'^* \restriction_{S_h}$
8 $\quad W \leftarrow W \setminus S_h$
9 $\quad R \leftarrow R - R_h$
10 $\vec{\rho}^* \restriction_W \leftarrow$ PoliteBinaryLambdaCrawl$(R, \vec{\mu} \restriction_W, \vec{\Delta} \restriction_W)$
11 Return $\vec{\rho}^*$

Algorithm 2, called PoliteBinaryLambdaCrawl, implements this intuition verbatim. Note that by the optimality of BinaryLambdaCrawl, *if* we are guaranteed that the optimal solution $\vec{\rho}^*$ to Problem 2 saturates the politeness constraint for a specific host h — allocating the full permissible crawl rate R_h to its set of pages S_h — we may readily obtain the optimal (in the sense of Problem 2) recrawl rates $\{\rho_i^* : i \in S_h\}$ via applying BinaryLambdaCrawl (Algorithm 1) to this host. Moreover, as remarked in [2] in the context of Problem 1 — and the same applies to the setting of Problem 2 after adding the politeness constraints — the objective function $F(\vec{\rho})$ is strictly concave on the convex polytope Q defined by the problem constraints. Consequently, there is a unique global maximum for the problem, and it is the only local extremum of the domain Q. Thus, there is a unique set of saturated politeness constraints corresponding to Problem 2's maximum. Hence, establishing PoliteBinaryLambdaCrawl's correctness amounts to proving that every politeness constraint R_h to which PoliteBinaryLambdaCrawl applies BinaryLambdaCrawl on line 6 of Algorithm 2 is in fact saturated by the optimal solution, so that we are justified in executing line 7.

3.3 Correctness

PoliteBinaryLambdaCrawl's key property is:

Theorem 3.2. *Algorithm 2 finds the optimal solution for Problem 2 in time $O(n \log n + mn)$.*

As described above, proving that repeatedly solving the subproblems on the saturated hosts as in Algorithm 2 leads to the (unique) maximizer $\vec{\rho}^*$ reduces to establishing the following lemma:

Lemma 3.3. *For every input $R, \vec{\mu}, \vec{\Delta}, \{(S_h, R_h)\}_{h=1}^m$ to Problem 2, let $\vec{\rho}^*$ be the maximizer of Problem 2 with this input, and let $\vec{\rho}'^*$ be the maximizer of Problem 1 with the input $R, \vec{\mu}, \vec{\Delta}$. If $\vec{\rho}'^*$ has $\sum_{i \in S_h} \rho_i'^* > R_h$ for some h (violating a politeness constraint), then $\vec{\rho}^*$ necessarily has $\sum_{i \in S_h} \rho_i^* = R_h$ (saturating this constraint).*

Proof. Observe first that the maximizer $\vec{\rho}'^*$ was the unique local extremum of F in the polytope Q_1 given by Problem 1. Thus, by the Lagrange multiplier method, the Lagrangian $\mathcal{L}_1(\vec{\rho}, \lambda) := F(\vec{\rho}) + \lambda(R - \sum_{k=1}^n \rho_k)$ must obey $\nabla \mathcal{L}_1(\vec{\rho}'^*, \lambda'^*) = 0$ for the optimal solution $(\vec{\rho}'^*, \lambda'^*)$. Computing the gradient of $\mathcal{L}_1$, we get $\frac{\partial}{\partial \rho_i} F \restriction_{\rho_i'^*} = \lambda'^* = \frac{\partial}{\partial \rho_j} F \restriction_{\rho_j'^*}$. For F defined in Eq. 2 this implies

$$\frac{\mu_i \Delta_i}{(\rho_i'^* + \Delta_i)^2} = \frac{\mu_j \Delta_j}{(\rho_j'^* + \Delta_j)^2} \quad \text{for all pages } i, j \in \{1, \ldots, n\}\,. \tag{5}$$

Next, consider the equivalent formulation of Problem 2 using Lagrange multipliers with the additional slack variables $\{q_1, \ldots, q_m\}$, where we rewrite the host politeness constraints as $\sum_{i \in S_h} \rho_i = R_h - q_h$ for $q_h \geq 0$. In this setting, we now have

$$\mathcal{L}_2(\vec{\rho}, \lambda, \lambda_1, \ldots, \lambda_m) := \mathcal{L}_1(\vec{\rho}, \lambda) + \sum_{h=1}^m \lambda_h (R_h - \sum_{i \in S_h} \rho_i)\,.$$

where the Karush-Kuhn-Tucker conditions dictate that $\lambda_h \geq 0$ for all h. By complementary slackness, we infer that $\lambda_h^* = 0$ for every h such that $q_h > 0$ in the optimal solution $(\vec{\rho}^*, \lambda^*, \lambda_1^*, \ldots, \lambda_m^*)$. For any pages $i \in S_h$ and $j \in S_\ell$ on hosts h and ℓ we obtain the identity $\frac{\partial}{\partial \rho_i^*} F \restriction_{\rho_i^*} - \lambda_h^* = \lambda^* = \frac{\partial}{\partial \rho_j} F \restriction_{\rho_j^*} - \lambda_\ell^*$, that is

$$\frac{\mu_i \Delta_i}{(\rho_i^* + \Delta_i)^2} - \lambda_h^* = \frac{\mu_j \Delta_j}{(\rho_j^* + \Delta_j)^2} - \lambda_\ell^* \quad \text{where } i, j \in \{1, \ldots, n\}\,. \tag{6}$$

For contradiction, suppose that the politeness-unconstrained solution $\vec{\rho}'^*$ satisfies $\sum_{i \in S_h} \rho_i'^* > R_h$ yet $\sum_{i \in S_h} \rho_i^* < R_h$. Since $\sum_{i=1}^n \rho_i^* = \sum_{i=1}^n \rho_i'^* = R$, there must exist some hosts $\ell \neq h$ such that $\sum_{j \in S_\ell} \rho_j'^* < \sum_{j \in S_\ell} \rho_j^*$. In particular, there must exist a page $i \in S_h$ and a page $j \in S_\ell$ such that

$$\rho_i^* < \rho_i'^* \quad \text{whereas} \quad \rho_j^* > \rho_j'^*\,. \tag{7}$$

However, all partial derivatives of F, given by the mapping $x \mapsto \mu\Delta/(x + \Delta)^2$, are monotone decreasing in $x \in [0, \infty)$. In addition, since we assumed that $\sum_{i \in S_h} \rho_i^* < R_h$, i.e., the optimal crawl rates for host h do *not* saturate h's politeness constraint, we must have $\lambda_h^* = 0$. Thus, for any $\vec{\mu}, \vec{\Delta} > 0$, Equations 6–7 imply that

$$\frac{\mu_i \Delta_i}{(\rho_i'^* + \Delta_i)^2} < \frac{\mu_i \Delta_i}{(\rho_i^* + \Delta_i)^2} \leq \frac{\mu_j \Delta_j}{(\rho_j^* + \Delta_j)^2} < \frac{\mu_j \Delta_j}{(\rho_j'^* + \Delta_j)^2}\,,$$

since $\lambda_h^* = 0 \leq \lambda_\ell^*$. This contradicts Equation 5 and thereby completes the proof. ∎

The time complexity of PoliteBinaryLambdaCrawl follows because it uses BinaryLambdaCrawl as a subroutine, and BinaryLambdaCrawl involves sorting all the pages, incurring the cost of $O(n \log n)$. However, this sorting step needs to be done only once during the first PoliteBinaryLambdaCrawl's iteration – subsequent calls to BinaryLambdaCrawl don't need to resort pages. Each iteration of PoliteBinaryLambdaCrawl either discovers a violated constraint, invoking BinaryLambdaCrawl and removing the corresponding host from subsequent iterations, or returns the final result. Thus, the complexity of iteration is $O(mn)$, and the total time complexity if $O(n \log n + mn)$.

3.4 Heuristic and Its Optimality

Using PoliteBinaryLambdaCrawl, as well as BinaryLambdaCrawl, in practice requires knowing pages' importance scores and change rates. Obtaining the change rates may be problematic, raising the question: can we approximate PoliteBinaryLambdaCrawl with another crawl scheduling method that doesn't rely on them?

Consider an algorithm for Problem 1 that assigns

$$\rho_i^{Imp} = \frac{R\mu_i}{\sum_{i'=1}^n \mu_{i'}}, \tag{8}$$

i.e., crawls the pages at rates proportional to their normalized importance. This algorithm is mentioned in the literature before (see, e.g., [8, 11]), and we call it *ImportanceCrawl*.

So far, ImportanceCrawl has been known as a mere heuristic. The following result, provides the first, to our knowledge, theoretical characterization of ImportanceCrawl's performance.

Proposition 1. *If the value ratio $\frac{\mu_i}{\Delta_i}$ is the same for all $i \in W$, ImportanceCrawl is optimal in the absence of politeness constraints.*

This follows immediately from the optimality of BinaryLambdaCrawl by plugging $\frac{\mu_i}{\Delta_i} = c$, $c > 0$ into BinaryLambdaCrawl's pseudocode (Algorithm 1). In this case, (a) BinaryLambdaCrawl assigns $\rho_i > 0$ to all pages, and (b) $\rho_i = \frac{R\mu_i}{\sum_{i=1}^n \mu_i}$ for all i, which is exactly the definition of ImportanceCrawl (Equation 8).

Furthermore, in the same way as PoliteBinaryLambdaCrawl's optimality is implied by the optimality of BinaryLambdaCrawl via Theorem 3.2, the following holds as well:

Proposition 2. *Consider the algorithm PoliteImportanceCrawl that results from replacing BinaryLambdaCrawl with ImportanceCrawl in PoliteBinaryLambdaCrawl (Algorithm 2). If the value ratio is the same for all pages i, PoliteImportanceCrawl is optimal under politeness constraints.*

Using the same reasoning, one can show that another popular freshness crawl heuristic that we call ChangeRateCrawl, which assigns recrawl rates in proportion to pages' change rates, [2, 8, 11]:

$$\rho_i^{ChR} = \frac{R\Delta_i}{\sum_{i'=1}^n \Delta_{i'}}, \tag{9}$$

has the same optimality guarantees. Thus, ImportanceCrawl and ChangeRateCrawl are equivalent for page sets with uniform value ratios.

Since PoliteImportanceCrawl is not only oblivious of page change rates but also doesn't involve PoliteBinaryLambdaCrawl's page sorting step, its conditional optimality property can make it a

preferred alternative to PoliteBinaryLambdaCrawl, as long as the value ratio statistics of the given page set are favorable. Our experiments explore this connection.

3.5 Implementation Considerations

Deploying randomized crawl-rate-based policies computed by PoliteBinaryLambdaCrawl (and other algorithms in this paper) involves an important caveat. Using a policy's crawl rates to sample a sequence of page crawl times from a Poisson process is sometimes not enough to guarantee observance of politeness constraints in practice. This is because hosts may require a time interval of a certain length to pass between successive URL requests. While such intervals imply politeness constraints expressed in terms of crawl rates, the converse is not true – sampling may result in a fast sequence of page requests violating the host's timeout requirements.

A solution to this problem is derandomizing the Poisson crawl rates into a sequence of crawl times, either including the required timeout between requests as a constraint during derandomization [22] or using methods that guarantee that the timeout will be observed with high probability [2, 14]. Azar et al. [2] also show that derandomization improves policy performance in practice.

Implementing a crawling policy in a real search engine entails other subproblems as well, such as deciding how to assign individual URL crawls to *fetchers*, crawler's machines or processes that actually download the content, so as to minimize synchronization overhead between them. This task can be handled using existing methods [13, 22]. In general, in light of such low-level considerations, the contribution of our work is an algorithm for computing a high-level politeness-compliant policy that meaningfully constrains the crawler's behavior but is sufficiently flexible to allow lower-level handling of Web crawling's intricacies.

4 RELATED WORK

Web crawling is a large research area, and we refer the reader to a survey by Olson and Najork [15] for a coverage of it. In particular, it discusses research such as [13] on another important aspect of crawl scheduling — the problem of assigning crawls of individual pages to crawl processes. Due to that problem's crawler architecture-specific nature, we do not touch upon it in this work.

To our knowledge, no prior work has proposed a method for computing an optimal crawl policy that takes into account both page importance and politeness constraints. However, refresh crawl policy optimization without politeness, under different optimization objectives and using different models for page change processes has been studied widely [2, 6–10, 16, 22]. Cho and Garcia-Molina [8] introduced the freshness-based optimization objective equivalent to ours as well as the Poisson process-based page change model that we use. However, they studied policies for this problem under the uniform page importance assumption. In particular, they show that in that case, the recrawl policy with rates proportional to page change rates can be wildly suboptimal, as hypothesized earlier by Coffman et al. [11]. For non-uniform page importance and crawls under the crawler's request rate constraint, an optimal policy computation algorithm has been proposed in [2].

There are several other freshness crawl optimization objectives and page change models that have also received attention in the literature. For instance, Pandey and Olston [19] propose taking into account the degree to which page changes affect search results, Olston and Pandey [16] study information longevity, and Wolf et al. [22] focus on an embarrassment minimization objective, where pages likely to be most clicked get recrawled more frequently, under several page change process types.

There has been little work on handling politeness constraints as part of crawl optimization. In the literature, they are handled at the level of crawler architecture, by appropriately throttling the stream of page requests to hosts [3, 21], although Wolf et al's model [22] could be adapted to handle a variant of them.

Our model assumes its parameters to be known. Importance is typically defined by the search engine itself based on available URL information[19, 22], including PageRank [17]. Page change rates can be learned from historical data, individually [9] or using page similarities [1, 10].

5 EMPIRICAL EVALUATION

The goal of our empirical evaluation was two-fold:

- *To explore the scalability of PoliteBinaryLambdaCrawl* in practice as a function of the number of URLs that need to be crawled and as a function of the number of politeness constraints that BinaryLambdaCrawl's unconstrained solution violates.
- *To assess the solution quatlity of PoliteBinaryLambdaCrawl* relative to politeness-respecting heuristic approaches, including PoliteImportanceCrawl.

In terms of scalability, we see that PoliteBinaryLambdaCrawl's theoretical time complexity is overly pessimistic. In reality, PoliteBinaryLambdaCrawl's running time depends on the number of politeness constraints very weakly, with only a few iterations (see Algorithm 2) of BinaryLambdaCrawlbeing enough to get the optimal constrained solution. In terms of solution quality, the results on an 18.5-million-URL dataset show that the tighter the politeness constraints are, the bigger advantage PoliteBinaryLambdaCrawl has over the alternatives. Before presenting the results in detail, we describe our experiment setup. The dataset and code used in the experiments are available at https://github.com/microsoft/MSMARCO-Optimal-Freshness-Crawl-Under-Politeness-Constraints.

5.1 Experiment Setup

Dataset. We collected a dataset by crawling 26,598,973 URLs for approximately 14 weeks using Microsoft Bing's production web crawler. For each of these URLs, we knew with high certainty whether the URL's host had a politeness constraint for Bing or not. We set the crawler to request each page approximately once a day, but occasionally crawls would fail for various reasons such as temporary host unavailability or spikes in the crawler's workload. This set of URLs is also crawled regularly, although not nearly as frequently, by the Bing search engine for the purpose of extracting structured information (time tables, event data, etc). On every crawl of a page, we applied Bing's information extraction templates to the page's content, and considered the page as changed from one crawl to the next if and only if the data extracted from its content differed across the two crawls. These templates filtered out page sections with highly changeable but not very meaningful content such as advertisements and *Related Links* lists; thus, our page change

detection procedure was robust to noise. We used Bing's importance scores μ_i for these pages (Figure 1), ranging from 0 to 65535.

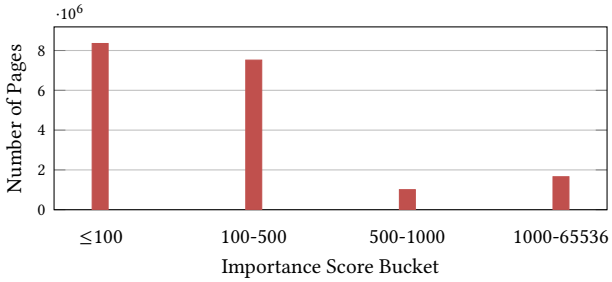

Figure 1: Importance score histogram for the 18,532,326 pages with strictly positive change rates in our dataset. Possible scores range from 1 to 65536. The distribution is very skewed, with most pages having importance less than 1000.

From the above data, we computed page change rate estimates $\{\hat{\Delta}_i\}_{i \in W}$ using the maximum-likelihood estimator for potentially incomplete page change history [9]. Indeed, in spite of almost daily recrawls, we could not guarantee that we observed every page change, i.e., that each page changed at most once between successive recrawls. The change rates were measured in changes per day. Out of the 26,598,973 URLs, 18,532,326 changed at least once during the 14-week observation period. These are the pages we used in the experiments.

Figures 1 and 2 show the characteristics of the resulting URL set. Figures 1 shows that most of the importance scores are below 1000 out of possible 65536. Only $\sim 1.6M$ URLs out of $18.5M$ have scores above 1000, i.e., significantly higher than average. Figure 2 describes the dataset property that, according to Proposition 2, determines the optimality of PoliteImportanceCrawl on it — the distribution of URL value ratios μ_i/Δ_i (see Section 3.4). Ranking URLs in the descending order of value ratio indicates that the value ratio can be approximated very well by a constant. As with importance scores, several hundred thousand URLs have markedly higher value ratios than the rest.

Host Politeness Constraints. The 18,532,326 URLs on which we evaluated policy quality came from 31599 hosts. Of these, only 1,688 hosts had politeness constraints, but they accounted for a

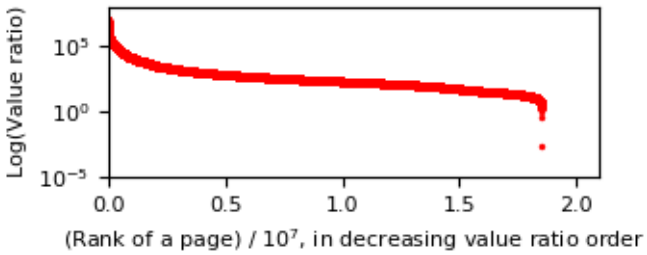

Figure 2: A ranking of 18,532,326 positive-change-rate URLs in the decreasing order of their *value ratios* μ_i/Δ_i (see Section 3.4). The value ratio distribution can be approximated well by a constant across most of the URLs, except the very head and tail of the ranking.

total of 17,144,883 of the above pages. Thus, just 1,387,443 URLs, less than 7% of the total, could theoretically be crawled without regard to hosts' request rate limitations. For each host that had it, its politeness constraint was measured in URL queries per day, like pages' change rates. In the experiments, we had 1688 simulated hosts, one for each actual host with a finite politeness constraint and one "virtual" host with an infinite politeness constraint that accounted for the remaining 1,387,443 pages.

Unfortunately, we couldn't use the hosts' real-world politeness constraints in the experiments. For each host, these constraints were designed with *all* URLs on that host in mind. However, our dataset included only a fraction of URLs from each host, and we didn't know the fraction of the host's total change rate for which our URLs accounted. Because of this, we couldn't scale down the known politeness constraints appropriately, and without scaling these constraints were far too permissive for our page set.

Instead, for each host h we defined its politeness constraint as $R_h = C_p \sum_{i \in S_h} \hat{\Delta}_i$, where $\sum_{i \in S_h} \hat{\Delta}_i$ is the total estimated change rate of h's URLs *in our dataset*, and C_p is a dataset-wide "politeness constraint coefficient". In the experiments, we varied C_p from 0.01 to 1 (see Figures 6 – 7), thereby varying constraints from very tight to relatively loose.

The grouping of URLs by host and flags indicating whether each host has a finite politeness constraint are available in the aforementioned dataset at https://github.com/microsoft/MSMARCO-Optimal-Freshness-Crawl-Under-Politeness-Constraints.

Crawl rate constraint. In the experiments, we use various subsets W' of URLs from the dataset described above. For every experiment with URL subset W', we fix the crawler's own request rate constraint to 20% of the total change rate of the URLs in W', i.e., $R = 0.2 \cdot \sum_{i \in W'} \hat{\Delta}_i$.

Algorithms and performance metric. We compared PoliteBinaryLambdaCrawl to two strong alternatives and one baseline:

- PoliteImportanceCrawl. As Section 3.4's theory implies, its performance can match PoliteBinaryLambdaCrawl's in certain settings. Our evaluation includes both these settings and situations where PoliteImportanceCrawl's optimality condition is violated to various degrees.
- RespectfulBinaryLambdaCrawl, a version of BinaryLambdaCrawl we designed for this experiment that abides by politeness constraints but at the cost of wasting crawl rate budget. Specifically, a current common practice in dealing with politeness constraints is adjusting the URL request rate at the level of the crawler's architecture as explained in [3, 21], to ensure it does not exceed relevant politeness constraints even if the crawler would like to contact a given host more frequently. RespectfulBinaryLambdaCrawl simulates this behavior. It uses BinaryLambdaCrawl to compute an optimal unconstrained policy, and then checks it against each host's politeness constraint. If this policy violates a constraint, RespectfulBinaryLambdaCrawl crawls that host at the constraint rate. However, in this case the excess rate allocated by BinaryLambdaCrawl to this host is wasted. Nonetheless, RespectfulBinaryLambdaCrawl is a good heuristic when politeness constraints are not very restrictive.

- PoliteUniformCrawl is an algorithm that serves as the baseline. It operates like PoliteBinaryLambdaCrawl but for each host with a violated constraint divides that host's constraint crawl rate equally among that host's URLs, i.e., uses uniform crawl rate distribution instead of BinaryLambdaCrawl or ImportanceCrawl in Algorithm 2.

Denote the set of these algorithms and PoliteBinaryLambdaCrawl as $\mathcal{A}$. We compute each algorithm $A \in \mathcal{A}$'s policy value F^A by plugging its output policy parameters $\vec{\rho}$ along with the learned change rates $\vec{\hat{\Delta}}$ and importance scores $\vec{\mu}$ from Figure 1 into Equation 2. However, our experiments are conducted on several datasets and under various politeness constraints, which makes policy values of the same algorithm on different datasets drastically different. To compare performance across datasets, we evaluate algorithms w.r.t. the performance of the baseline, PoliteUniformCrawl, by comparing algorithms' relative scores F^A_{rel}:

$$F^A_{rel} = \frac{F^A - F^{base}}{\max_{A' \in \mathcal{A}} F^{A'} - F^{base}}$$

Here, F^A is algorithm A's absolute policy value in a given experiment, F^{base} is PoliteUniformCrawl's performance there, and $\max_{A' \in \mathcal{A}} F^{A'}$ is the maximum absolute policy value of any algorithm participating in the experiment. Clearly, for any algorithm, $F^A_{rel} \leq 1$. Also in our experiments F^A_{rel} was always non-negative, because PoliteUniformCrawl (F^{base}) happened to be the weakest performer in every experiment.

Software and hardware. We implemented all algorithms in Python 3.7 and ran them on a Windows 10 laptop with 16GB RAM and an Intel quad-core 2.11GHz i7-8650U CPU.

5.2 Empirical Results

Scalability. We begin studying PoliteBinaryLambdaCrawl's performance by examining its scalability as a function of the number of politeness constraints. Theory predicts PoliteBinaryLambdaCrawl's number of iterations to scale linearly in the constraint number, which could be prohibitively expensive in practice. Fortunately, as Figure 3 suggests, in reality PoliteBinaryLambdaCrawl makes no more than 6 iterations on our dataset with 1688 constraints, even in very tightly constrained scenarios with coefficient $C_p = 0.01$. A possible reason for this is that PoliteBinaryLambdaCrawl's running time depends on the number of constraints PoliteBinaryLambdaCrawl *activates per iteration* — and hence on the number of pages for which PoliteBinaryLambdaCrawl computes final crawl rate values per iteration – rather than purely on the number of constraints a problem instance has. Figure 3 shows that if constraints are very tight (low C_p), then PoliteBinaryLambdaCrawl is forced to find final crawl rate values for many URLs in the first iteration, thereby making fast progress and solving the problem after just a few iterations. If constraints are loose (high C_p), then the unconstrained solutions computed by BinaryLambdaCrawl in each iteration violate few of them, again ensuring that the optimal constrained solution is reached after few iterations.

Figure 4 shows PoliteBinaryLambdaCrawl's solution time as a function of page set size, which varies from 100,000 to 18,532,326, and the degree to which the problem is constrained, as measured by

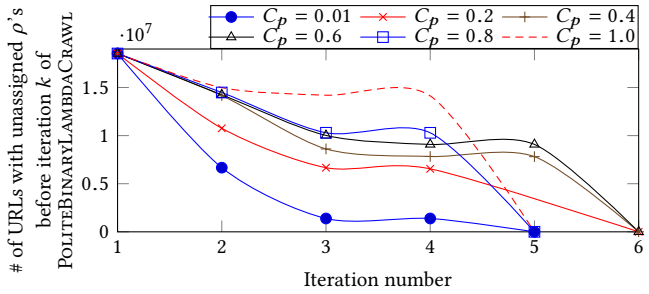

Figure 3: Number of URLs whose crawl rates weren't finalized before k-th iteration of PoliteBinaryLambdaCrawl, for politeness constraint coefficients C_p ranging from 0.01 to 1.0 on 18.5M URLs and 1688 hosts. Small C_p values indicate tight constraints. They cause BinaryLambdaCrawl's unconstrained solution to violate many hosts' constraints in iteration 1 of PoliteBinaryLambdaCrawl. Accordingly, PoliteBinaryLambdaCrawl solves for the correctly constrained crawl rates for these hosts' URLs in the same iteration, leaving fewer URLs for subsequent iterations. Thus, smaller C_p values yield "lower" plots.

C_p varying from 0.01 to 1. The datasets used in this experiment are "prefixes" of the full dataset sorted in the decreasing order of URL value ratios. These plots suggest PoliteBinaryLambdaCrawl's nearly linear scalability in the number of pages. They also show that C_p non-negligibly affects execution time.

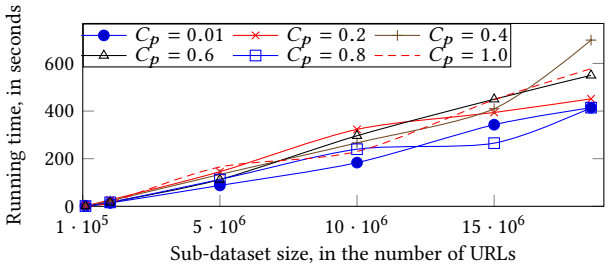

Figure 4: PoliteBinaryLambdaCrawl's execution time for prefixes of the full dataset from 100,000 to 18,532,326 pages, and constraint coefficients C_p from 0.01 to 1. PoliteBinaryLambdaCrawl scales linearly with dataset sizes in this range, but C_p affects the running time significantly.

Figure 5 gives additional insight into this dependence, varying C_p on the full dataset of 18.5M URLs. It implies that PoliteBinaryLambdaCrawl's dependence on constraint tightness is not only significant but also non-monotonic: the $C_p = 0.4$ instance is harder to solve than $C_p = 0.2$ and $C_p = 0.6$.

Policy quality comparison. This set of experiments aims to answer the question: under what circumstances does PoliteBinaryLambdaCrawl have a significant policy quality advantage over strong heuristics, and when can these computationally cheaper heuristics replace it? The first experiment, whose results are in Figure 6, compares PoliteBinaryLambdaCrawl, PoliteImportanceCrawl, and RespectfulBinaryLambdaCrawl on the full

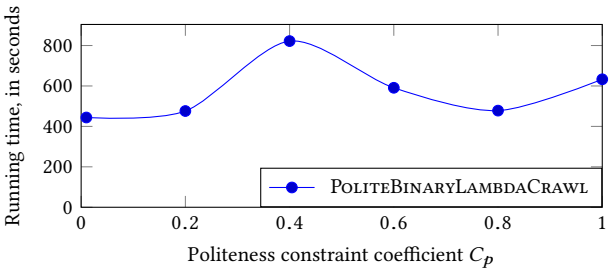

Figure 5: PoliteBinaryLambdaCrawl's execution time as a function of C_p on the full dataset. The dependence is complex and non-monotonic, which is indirectly corroborated by Figure 3.

dataset for different degrees of constraint tightness C_p. When constraints are tight, PoliteBinaryLambdaCrawl outperforms the second-best alternative, PoliteImportanceCrawl, by a large margin, but its advantage dwindles as constraints become looser. In particular, RespectfulBinaryLambdaCrawl in addition to PoliteImportanceCrawl becomes a viable option. This is to be expected: when politeness constraints are loose, the optimal *unconstrained* solution produced by BinaryLambdaCrawl violates only a few constraints, so RespectfulBinaryLambdaCrawl doesn't waste too much crawl bandwidth when adapting this solution to fit the constraints perfectly.

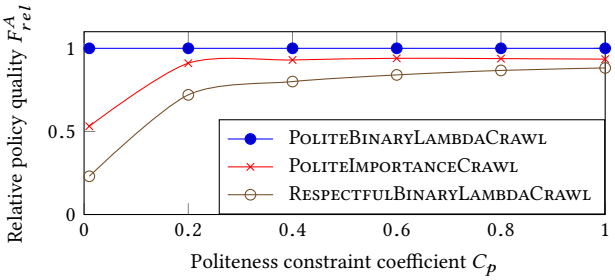

Figure 6: Relative policy quality of three algorithms on the full dataset (18.5M URLs) for constraint coefficient C_p varying from 0.01 to 1. PoliteUniformCrawl's performance is the weakest for all values of C_p, so its policy score F^A_{rel} is always 0 and we have omitted its plot for clarity. PoliteBinaryLambdaCrawl's advantage is greatest in tightly constrained scenarios (low C_p). As politeness constraints become less restrictive, PoliteUniformCrawl's and RespectfulBinaryLambdaCrawl's performance predictably converges.

We hypothesize that PoliteImportanceCrawl performs well on our dataset because, as Figure 2 demonstrates, its value ratio distribution is fairly uniform. To test this hypothesis and assess PoliteImportanceCrawl's performance when this assumption is violated, we ran PoliteBinaryLambdaCrawl and PoliteImportanceCrawl on top n pages from our dataset ranked as in Figure 2, with n ranging from 100,000 to 18,532,326. Note that for low values of n, the dataset prefixes have very non-uniform value ratio distributions, which should hurt PoliteImportanceCrawl's performance. Figure 7 shows that this is exactly what happens in reality. In this experiment, we fix $C_p = 0.2$ — a constraint coefficient for which PoliteImportanceCrawl's policy on the full dataset is only ∼ 10% worse than PoliteBinaryLambdaCrawl's (Figure 6). On the 100,000-URL dataset prefix, however, PoliteImportanceCrawl's solution turns out to be 50% worse. At the same time, as the prefix grows to sizes over 5,000,000, where pages from the uniform middle of the value ratio distribution dominate, PoliteImportanceCrawl's solution quality rebounds.

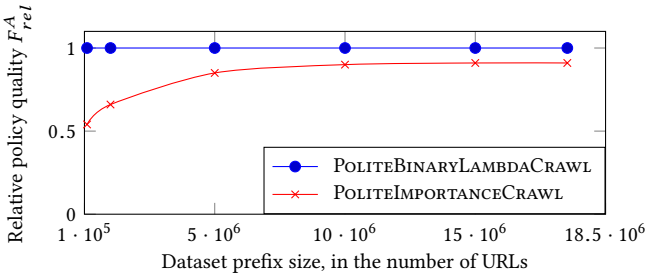

Figure 7: PoliteBinaryLambdaCrawl vs PoliteImportanceCrawl on various prefixes of the full dataset sorted in the decreasing order of value ratio (see Figure 2), with constraint coefficient fixed at $C_p = 0.2$. Prefix sizes vary from 100,000 to 18,532,326 URLs. As Figure 1 shows, prefixes of size 100,000, 1,000,000, and 5,000,000 have very non-uniform value ratio distributions, which hurts PoliteImportanceCrawl's performance. For prefix sizes past 5,000,000, the value ratio distribution is more uniform and PoliteImportanceCrawl's performance approaches PoliteBinaryLambdaCrawl's, as theory predicts.

6 CONCLUSION

This paper presented PoliteBinaryLambdaCrawl, the first optimal algorithm for scheduling freshness crawl under the crawler' request rate and hosts' politeness constraints for pages with non-uniform importance and change rates. PoliteBinaryLambdaCrawl' central idea is an iterative reduction of this problem to a set of small instances of crawl scheduling under a crawl rate constraint alone. Empirically, PoliteBinaryLambdaCrawl's biggest advantage over alternatives is in settings with tight politeness constraints and page sets with highly non-uniform value ratios.

We also introduced PoliteImportanceCrawl, an approximation algorithm that doesn't use change rates for scheduling. Applying it in practice calls for verifying its uniform-value-ratio optimality condition, which still involves measuring page change rates but can be done on a small sample of pages. Even if value ratio uniformity doesn't hold exactly, our experiments show that it performs well. This suggests a hybrid PoliteBinaryLambdaCrawl-PoliteImportanceCrawl approach, whereby PoliteBinaryLambdaCrawl uses a very limited crawl budget to revisit pages with high value ratios, and PoliteImportanceCrawl schedules crawls for the rest. Evaluating this idea is a topic for future investigation.

ACKNOWLEDGMENTS

We would like to thank Cheng Lu, Arnold Overwijk, Fabrice Canel, Junaid Ahmed, and Daniel Campos from Microsoft Bing for useful discussions and help with collecting data for this paper's empirical evaluation.

REFERENCES

[1] 2013. Predicting content change on the web. In *WSDM*. 415–424.
[2] Y. Azar, E. Horvitz, E. Lubetzky, Y. Peres, and D. Shahaf. 2018. Tractable Near-optimal Policies for Crawling. *Proceedings of the National Academy of Sciences (PNAS)* (2018).
[3] P. Boldi, A. Marino, M. Santini, and S. Vigna. 2014. BUbiNG: massive crawling for the masses. In *WWW Companion*.
[4] B. Brewington and G. Cybenko. 2000. How dynamic is the Web. In *WWW*.
[5] L. Bright, A. Gal, and L. Raschid. 2006. Adaptive Pull-Based Policies for Wide Area Data Delivery. *ACM Transactions on Database Systems (TODS)* 31, 2 (2006), 631–671.
[6] J. Cho and H. Garcia-Molina. 2000. The evolution of the web and implications for an incremental crawler.
[7] J. Cho and H. Garcia-Molina. 2000. Synchronizing a Database to Improve Freshness. In *ACM SIGMOD International Conference on Management of Data*.
[8] J. Cho and H. Garcia-Molina. 2003. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems* 28, 4 (2003), 390–426.
[9] J. Cho and H. Garcia-Molina. 2003. Estimating frequency of change. *ACM Transactions on Internet Technology* 3, 3 (2003), 256–290.
[10] J. Cho and A. Ntoulas. 2002. Effective change detection using sampling. In *VLDB*.
[11] E. G. Coffman, Z. Liu, and R. R. Weber. 1998. Optimal robot scheduling for web search engines. *Journal of Scheduling* 1, 1 (1998).
[12] J. Eckstein, A. Gal, and S. Reiner. 2007. Monitoring an Information Source Under a Politeness Constraint. *INFORMS Journal on Computing* 20, 1 (2007), 3–20.
[13] J. Edwards, K. S.McCurley, and J. A. Tomlin. 2001. An adaptive model for optimizing performance of an incremental web crawler. In *WWW*.
[14] N. Immorlica and R. Kleinberg. 2018. Recharging bandits. In *FOCS*.
[15] C. Olston and M. Najork. 2010. Web Crawling. *Foundations and Trends in Information Retrieval* 3, 1 (2010), 175–246.
[16] C. Olston and Sandeep Pandey. 2008. Recrawl scheduling based on information longevity. In *WWW*. 437–446.
[17] L. Page, S. Brin, R. Motwani, and T. Winograd. 1998. *The PageRank citation ranking: Bringing order to the web.* Technical Report. MA, USA.
[18] S. Pandey, K. Dhamdhere, and C. Olston. 2004. WIC: A general-purpose algorithm for monitoring Web information sources.
[19] S. Pandey and C. Olston. 2005. User-centric web crawling. In *WWW*.
[20] R. Rashkovits and A. Gal. 2013. A Cooperative Model for Preference-Based Information Sharing in Narrow Bandwidth Networks. *International Journal of Cooperative Information Systems* 22, 10 (2013).
[21] V. Shkapenyuk and T. Suel. 2002. Design and implementation of a high performance distributed web crawler. In *ICDE*.
[22] J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. 2002. Optimal crawling strategies for web search engines. In *WWW*.