Improving the Exploration/Exploitation Trade-Off in Web Content Discovery

Peter Schulam
schulamp@amazon.com
Amazon Alexa
USA

Ion Muslea
musleaim@amazon.com
Amazon Alexa
USA

ABSTRACT

New web content is published constantly, and although protocols such as RSS can notify subscribers of new pages, they are not always implemented or actively maintained. A more reliable way to discover new content is to periodically re-crawl the target sites. Designing such "content discovery crawlers" has important applications, for example, in web search, digital assistants, business, humanitarian aid, and law enforcement. Existing approaches assume that each site of interest has a relatively small set of unknown "source pages" that, when refreshed, frequently provide hyperlinks to the majority of new content. The state of the art (SOTA) uses ideas from the multi-armed bandit literature to explore candidate sources while simultaneously exploiting known good sources. We observe, however, that the SOTA uses a sub-optimal algorithm for balancing exploration and exploitation. We trace this back to a mismatch between the space of actions that the SOTA algorithm models and the space of actions that the crawler must actually choose from. Our proposed approach, the Thompson crawler (named after the Thompson sampler that drives its refresh decisions), addresses this shortcoming by more faithfully modeling the action space. On a dataset of 4,070 source pages drawn from 53 news domains over a period of 7 weeks, we show that, on average, the Thompson crawler discovers 20% more new pages, finds pages 6 hours earlier, and uses 14 fewer refreshes per 100 pages discovered than the SOTA.

CCS CONCEPTS

• **Information systems** → **Web crawling**; • **Computing methodologies** → **Online learning settings**.

KEYWORDS

Web Content Discovery, Multi-Armed Bandits, Thompson Sampling

ACM Reference Format:
Peter Schulam and Ion Muslea. 2023. Improving the Exploration/Exploitation Trade-Off in Web Content Discovery. In *Companion Proceedings of the ACM Web Conference 2023 (WWW '23 Companion), April 30–May 04, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3543873.3587574

WWW '23 Companion, April 30–May 04, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9419-2/23/04.
https://doi.org/10.1145/3543873.3587574

1 INTRODUCTION

Many applications built on the web depend on real-time access to newly published content. For instance, people expect to read breaking news on their phones or tablets, businesses require up-to-date information on competitors' offerings, and humanitarian aid organizations use the web to monitor developing crises. Although protocols such as sitemaps and RSS feeds (see e.g. [20]) notify subscribers of new content, they are not always available or actively maintained. A more reliable way to quickly discover new content is to periodically re-crawl target sites [10].

There are two high-level approaches to re-crawling a site in order to discover new content. First, we can periodically crawl the site exhaustively (see e.g. Olston and Najork [15]). Although this approach often discovers a large amount of the available content (we do not consider the problem of discovering "deep" content in this paper [3]), it can be prohibitively expensive and slow as a means to discover pages in near real-time (e.g. within minutes of publication). Moreover, there are rising concerns about the impact that large crawling operations can have on carbon emissions [12, 21], making it increasingly important to reduce the *overhead* of discovering new content [10]; i.e. the number of known pages we must refresh per new page discovered.

The second, more precise, approach to discovering new content on a target site is to periodically refresh a collection of *source pages* (e.g. the sports, business, and politics "landing pages" for a newspaper's website). Using weekly deep crawls of 200 sites, Dasgupta et al. [10] showed that the majority of new content is hyperlinked from a relatively small number of existing pages. The challenge, however, is to intelligently select the subset of pages that we will refresh. This is the problem of predicting the "yield" of a page; i.e. how many hyperlinks to new, unknown pages will we discover?

Contributions. The state of the art in near real-time content discovery [17] refreshes a fixed collection of *candidate* source pages (i.e. not all source pages will have high yield), and uses ideas from the multi-armed bandit literature [22] to simultaneously explore the candidate sources and exploit known good sources to discover new content. In this paper, we show that the method proposed in Pham et al. [17] (which we refer to as the "Pham crawler" from here on) uses a multi-armed bandit to model an action space that does not match the space of actions that the crawler must actually choose from. This mismatch leads to a sub-optimal trade-off between exploration and exploitation for the purpose of learning to predict page yield. This reduces the Pham crawler's coverage (number of new pages discovered), makes it less efficient (requires more refreshes per discovered page), and increases the amount of time required for it to discover a new page. To address this issue,

we propose the Thompson crawler (named for the Thompson sampler that drives its refresh decisions), which more faithfully models the action space. On a dataset of 4,070 source pages from 53 news domains over a period of 7 weeks, we show that, on average, the Thompson crawler discovers 20% more new pages, finds pages 6 hours earlier, and uses 14 fewer refreshes per 100 pages discovered than the Pham crawler.

1.1 Related Work

Dasgupta et al. [10] were the first to study the problem of discovering new pages by scheduling refreshes of known pages. To select pages to refresh, Dasgupta et al. [10] proposed a solution that interleaves low frequency "snapshot" crawls and higher frequency "discovery" crawls. A snapshot crawl exhaustively crawls the site, while a discovery crawl refreshes a relatively small subset of known pages with the goal of discovering hyperlinks to unknown pages. Dasgupta et al. [10] compare the results of snapshot crawls over time to estimate the yield of known pages, and then refresh the pages with the highest estimated yield during the subsequent discovery crawls. Although this technique is effective, snapshot crawls can be prohibitively expensive for larger sites.

Pham et al. [17] recently proposed a technique that does not require snapshot crawls, instead learning directly from cheaper, more frequent discovery crawls. Their method is rooted in key ideas from the literature on multi-armed bandits, where a decision-making system simultaneously learns about its environment (exploration), while also using its current knowledge to take valuable actions (exploitation). They show that their crawler outperforms variants of the methods proposed in Dasgupta et al. [10] adapted to fit within a fixed budget per crawl cycle. Our work builds on their ideas.

There are several other notable works that address problems similar to the one we study here, but make different assumptions. Gupta et al. [11] also aim to selectively refresh pages with the highest expected yield. They design an algorithm, however, that operates in two distinct phases: a training phase where data is collected for the purpose of training a predictive model, and a crawling phase where the model's predictions drive what pages to refresh. In our setting, however, we wish to simultaneously learn a good predictive model of yield while also discovering new content. Lefortier et al. [14] study the problem of scheduling when to refresh a set of known good source pages and when to fetch the discovered content. We do not assume that all candidate source pages are good, however, and so must learn to focus on the most productive ones.

In the broader literature on crawling, a lot of research has focused on the related problem of keeping a fixed collection of web pages up-to-date [7–9, 13, 16]. Many of the policies used to drive refresh decisions depend on statistical models of how web pages change over time, which ties into a more general thread of modeling web dynamics independent of its application to crawling [1, 4, 5, 18].

2 METHODS

In this section, we formalize the problem of discovering new content from a fixed collection of candidate source pages, provide relevant background on algorithms for the multi-armed bandit problem, discuss our observed shortcoming of the Pham crawler, and propose the Thompson crawler as an alternative that addresses this shortcoming.

2.1 Fixed Source Content Discovery

We study the problem of discovering hyperlinks to new, unknown pages in discrete-time using a fixed set of candidate source pages. Formally, we define the study period $[1, T]$ where each time step t represents an hour of clock time (e.g. `2022-11-13 00:00:00` to `2022-11-13 01:00:00`, right side excluded). Over the course of the study period, a crawler monitors a fixed collection of *sources* $\mathcal{U}$, which it can refresh to discover new content. At each time t, the crawler is allowed a budget $k < |\mathcal{U}|$ of page refreshes. We denote the set of k pages refreshed at time t as $\mathcal{F}_t$. After refreshing the sources $\mathcal{F}_t$, we observe a set of *outlinks* from the source pages to *target* pages. We denote each outlink using a tuple (t, u, v), which indicates that we observe a link from source u to target v after refreshing the source at time t. We use $\mathcal{L}_{tu}$ to denote the unique set of outlink tuples observed from source u at time t.

At any time t, we define the target history $\mathcal{V}_t$ as the union of all targets observed in outlinks at times $t' < t$: i.e. $\mathcal{V}_t \triangleq \cup_{t'=1}^{t-1} \cup_{u \in \mathcal{F}_{t'}} \mathcal{L}_{t'u}$. At time $t = 1$, the target history $\mathcal{V}_1$ is the empty set. We define the *yield* y_{tu} of a source u at time t as the number of outlinks from u at time t that point to targets v that are not in the outlink history $\mathcal{V}_t$. The same target can count towards the yield of multiple sources at a given time step (i.e. more than one source can get "credit" for discovering a novel target). We define the crawler's yield y_t at time t as the number of target pages linked to by any source $u \in \mathcal{F}_t$ that is not in the history $\mathcal{V}_t$.

We use three metrics to evaluate a crawler's performance over the study period $[1, T]$: *coverage*, *overhead*, and *90th percentile of hours to discovery* (denoted HTD-P90). Let $\mathcal{V}^*$ denote the set of all targets that we can discover with any outlink from any source $u \in \mathcal{U}$ at any time $t \in [1, T]$. This set of targets represents our "ground truth" collection of pages that we would like to discover. The coverage of a crawler is measured as $|\mathcal{V}_{T+1} \cap \mathcal{V}^*||\mathcal{V}^*|^{-1}$ (where $|\cdot|$ denotes set cardinality). The overhead of a crawler is measured as $\sum_{t=1}^{T} |\mathcal{F}_t||\mathcal{V}_{T+1}|^{-1}$. Finally, HTD-P90 is the 90th percentile of the difference between when a target page first *appeared* (i.e. the earliest time when it was first hyperlinked from any source page) and when it was discovered by the crawler (i.e. the first time it was linked from a source page that the crawler chose to refresh).

2.2 Multi-Armed Bandits

The multi-armed bandit literature addresses the problem of how an agent can simultaneously operate successfully in an environment while continuing to learn about it. The problem is typically formalized using a collection of random variables $\{Y_a : a \in \mathcal{A}\}$, where $\mathcal{A}$ is a set of actions and Y_a models the unknown distribution over *rewards* that the agent receives. The agent operates over a period of discrete time, and at each time t must choose an action $a \in \mathcal{A}$. After selecting an action, the agent receives a reward y_a drawn from the distribution of the random variable Y_a.

If the agent knows the expected reward of each action, then the optimal strategy is to repeatedly choose the action with largest expected value. The challenge, however, is that the agent starts

with minimal knowledge of the reward distributions and must simultaneously estimate the expected rewards (explore), while also maximizing the cumulative reward received over time (exploit). Multi-armed bandit algorithms aim to trade off exploration and exploitation in a way that minimizes the agent's *regret*; the difference between the cumulative reward achieved using the optimal strategy and the cumulative reward that the agent actually obtains.

2.2.1 The UCB1 Algorithm. There is a massive literature on multi-armed bandits, but only two algorithms are relevant to our discussion. The first algorithm is the Upper Confidence Bound 1 (UCB1) algorithm, which was first proposed and analyzed by Auer et al. [2]. In the UCB1 algorithm, the agent keeps track of two statistics per arm: (1) the sample average of rewards $\hat{y}_a$ observed after taking action a, and (2) the number of times n_a that the agent has taken action a. The agent first chooses each action once to initialize statistics, and then at each subsequent time t chooses

$$a_t = \arg\max_{a \in \mathcal{A}} \left(\hat{y}_a + \sqrt{\frac{2\log(t-1)}{n_a}} \right) \tag{1}$$

The square root term in the expression above captures uncertainty in the agent's current estimates of the expected rewards. Choosing actions based on estimated upper bounds of expected rewards encourages the agent to explore actions that may prove more valuable than the current "greedy" action (i.e. the action with largest $\hat{y}_a$).

2.2.2 Thompson Sampling. The second multi-armed bandit algorithm relevant to the results in this paper is Thompson sampling. Thompson sampling is a *family* of algorithms built on the Bayesian approach to statistical estimation [23, 24]. As in the UCB1 algorithm, our goal is to estimate the expected reward of each action, but in the Thompson sampling approach we achieve this by computing the following posterior distribution at each time t:

$$p(\theta \mid y_{1:(t-1)}, a_{1:(t-1)}) \propto p(\theta) \prod_{i=1}^{t-1} p(y_{a_i} \mid a_i, \theta), \tag{2}$$

where $p(y_i \mid a_i, \theta)$ is a statistical model of the distribution for the reward Y_a associated with action a parameterized by θ, and $p(\theta)$ is a prior distribution over the reward distribution parameters. To select an action at time t, we draw a sample $\hat{\theta}_t$ from the posterior distribution above, and choose the action

$$a_t = \arg\max_{a \in \mathcal{A}} \mathbb{E}\left[Y_t \mid a, \hat{\theta}_t \right]. \tag{3}$$

In words: we sample a plausible configuration of parameters θ from the posterior, compute the expected value of each action as if our sample were the true parameters, and then select the action that maximizes the expected reward. In the interest of space, we omit a more extensive introduction to the topic, but refer interested readers to Russo et al. [19] for more on the history of Thompson sampling and a readable, tutorial-style introduction to the algorithm.

2.3 Pham Crawler

The discovery-only crawler proposed in Pham et al. [17] has two key components. The first component is a predictive model of the yield that the crawler will observe at time t for source u. The second component is an implementation of the UCB1 algorithm that selects what fraction of the total budget k at each time step should be allocated to exploitation and to exploration. We show that this architecture is sub-optimal for driving discovery-only crawls.

The expected yield model is a linear regression that includes a collection of 35 features for each source u. The 35 features include four summary statistics of the historical fetches of the source (average yield, standard deviation of yield, time since last fetch, and the product of average yield and age), and 31 features that depend on the time of the fetch (24 one-hot encoded features for hour of day, and 7 one-hot encoded features for day of week). At each time t, the crawler uses the model to predict the expected yield of each source and ranks the sources in descending order of the predicted yield. We emphasize that the crawler uses these predictions (and the induced rank over sources) to choose its actions (i.e. which sources to refresh at each time t).

The UCB1 component decides how much of the budget k to use for exploration and how much to use for exploitation. The "actions" that it models are the fraction of the budget to use for exploitation. Pham et al. [17] choose actions $[0.6, 0.7, 0.8, 0.9, 1.0]$ in their experiments. For a given action a, the crawler uses $k \times a$ of the budget to select the sources with the top predicted yields, and uses $k \times (1 - a)$ of the budget to select the most stale sources (i.e. those that have not been refreshed for the longest amount of time). For each of the five actions, the UCB1 component tracks the average combined yield $\hat{y}_a$ from all time steps when that action was selected.

We claim that the exploration/exploitation trade-off is sub-optimal in this architecture because the UCB1 algorithm tracks the expected yields and uncertainties of an action space that does not match the actions that the crawler actually chooses from. The crawler's true action space is comprised of the source pages $\mathcal{U}$, from which it selects a subset of of pages to refresh in order to maximize yield. The UCB1 component of the Pham crawler, however, does not track how certain the yield estimate model is about its predictions for each source. As a result, the UCB1 algorithm cannot use this information to precisely select which sources should be confidently refreshed (exploitation) and which sources have high potential, but are currently underexplored (exploration).

2.4 Thompson Crawler

To more faithfully model the actions of a discovery-only crawler, we introduce the Thompson crawler (named after Thompson sampling, which we use to balance exploration and exploitation). We posit a probabilistic model of page yield for each source at each hour of the day (i.e. `00`, `01`, ..., `23`), but our approach is straightforward to extend to more elaborate methods of partitioning time (e.g. modeling page yield for each hour of the week). We assume that page yield for source u at hour h is drawn from a Poisson distribution with rate parameter λ_{uh}, which places non-zero probability on all non-negative integers. We further assume that λ_{uh} for all sources $u \in \mathcal{U}$ and all hours $h \in \{0, \ldots, 23\}$ are sampled independently from a common gamma distribution with fixed hyperparameters α and β. In summary, we assume the following likelihood and prior

```python
import numpy as np
from scipy.stats import gamma

def thompson_crawl(n_src, n_hrs, tsps, k,
                   alpha=1.0, beta=1.0):

    s = np.zeros((n_src, n_hrs))
    n = np.zeros((n_src, n_hrs))

    for tsp in tsps:
        h = tsp_to_hour(tsp)
        a = s[:, h] + alpha
        b = n[:, h] + beta

        y_hat = gamma.rvs(a, scale=1.0/b)
        srcs = np.argsort(y_hat)[-k:]
        y_obs = fetch_and_calc_yields(srcs)

        for u, y in zip(srcs, y_obs):
            s[u, h] += y
            n[u, h] += 1
```

Figure 1: Thompson crawler in Python.

for each source u and hour h:

$$p(\lambda \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\beta\lambda} \tag{4}$$

$$p(y \mid \lambda) = \frac{1}{y!} \lambda^y e^{-\lambda} \tag{5}$$

This prior-likelihood combination is computationally convenient because the prior is *conjugate* to the likelihood. This means that the posterior distribution over λ given observed data y is also a gamma distribution. We can see this for the gamma-Poisson pair by writing out the posterior distribution

$$p(\lambda \mid y, \alpha, \beta) \propto p(y \mid \lambda) p(\lambda \mid, \alpha, \beta) \propto \lambda^y e^{-\lambda} \lambda^{\alpha-1} e^{-\beta\lambda} \tag{6}$$

$$= \lambda^{y+\alpha-1} e^{-(\beta+1)\lambda} \tag{7}$$

The final line on the right-hand-side has the same unnormalized form as a gamma distribution with parameters $\alpha' = x + \alpha$ and $\beta' = 1 + \beta$. We simply divide by $\frac{\Gamma(\alpha')}{\beta'^{\alpha'}}$ to obtain the posterior density over λ. For a collection of n observations $y_{1:n}$, we obtain a similar result where $\alpha' = \sum_{i=1}^n y_i + \alpha$ and $\beta' = n + \beta$. To maintain the posterior distribution over λ, it is sufficient to keep track of the sum of all observations $s = \sum_{i=1}^n y_i$ and the total number of observations n.

Figure 1 displays a simplified implementation of our Thompson crawler in Python. Note that the Thompson crawler tracks the same statistics per action as the UCB1 algorithm. We chose Thompson sampling over UCB1 for selecting actions based on these statistics because Thompson sampling has been shown to perform better in practice [6]. Our model of page yields is especially simple to implement, but we note that the Thompson crawler approach can be readily extended with more elaborate priors and likelihoods. The price to pay is (typically) more difficult posterior inference. For some production systems, this price may be justified, but we leave this direction open for future work.

3 EXPERIMENTS

3.1 Data

We collect data for our experiments by refreshing pages from a wide variety of news domains multiple times per hour.[1] We collect pages from 100 candidate news domains between November 13, 2022 and December 31, 2022 (inclusive; 1,176 total hours). We count the number of times each page is fetched over this 7-week period, and retain only the top 100 most frequently refreshed pages for each domain. We discretize the fetches to an hourly time grid by randomly sampling one fetch per page per hour. From this candidate set of pages, we further filter pages that do not have at least one fetch for at least 90% of the hours in the study period (i.e. we require at least 1,059 fetches for each page). We enforce this "density" requirement on the number of fetches per source so that we can reliably simulate a crawl retrospectively. Finally, we remove domains that have fewer than 40 total pages with one fetch in at least 90% of the hours in the study period (40 is twice the maximum budget k that we use in our experiments below). The final dataset contains 4,070 unique source pages from 53 news domains, 4.7M total fetches, and 589M total outlinks[2] to 1.1M unique target pages. We run each crawler independently on each domain using per-domain budgets $k = 5$, $k = 10$, and $k = 20$ (i.e. a total of three runs per crawler per domain).

3.2 Crawlers

We compare the performance of three crawlers on our retrospective dataset. In addition to the Pham crawler and the Thompson crawler, we also benchmark an *oracle* crawler.

3.2.1 Oracle crawler. The Oracle crawler selects the k pages to refresh at each time t using perfect knowledge of the outlinks that will appear on each source page. It uses the Greedy algorithm in Dasgupta et al. [10] to select k source pages in order to maximize the number of novel target pages that it will discover after fetching those pages.

3.2.2 Pham crawler. We implement the Bandit algorithm as described in Section 5.3 of Pham et al. [17] (and summarized in Section 2.3 above). We do not model overlap in targets between sources when selecting which pages to refresh.[3] We otherwise configure the crawler as reported by Pham et al. [17]: we retrain the yield model very 3 hours using data from the past 7 days and use a 24-hour window for computing the history-dependent features.

3.2.3 Thompson crawler. Finally, we implement the Thompson crawler as shown in Figure 1. We set $\alpha = 1$ and $\beta = 1$ for all

[1] We crawl politely by respecting `robots.txt` and by waiting at least several seconds between consecutive HTTP requests to the same domain.

[2] i.e. (t, u, v) tuples representing a link from source u to target v at time t

[3] Pham et al. [17] state that "we need to estimate the overlap differently" than Dasgupta et al. [10], but omit important details required to reproduce their method. Specifically, they do not provide an explicit algorithm for how the overlap estimates are updated with the k fetched pages. Dasgupta et al. [10] observed similar performance with and without accounting for overlap (compare Cliq-Win and Od-Win in their Table 2). Pham et al. [17] did not study the effect of modeling overlap with an ablation analysis, so we expect that it had little impact on their results (as observed by [10]).

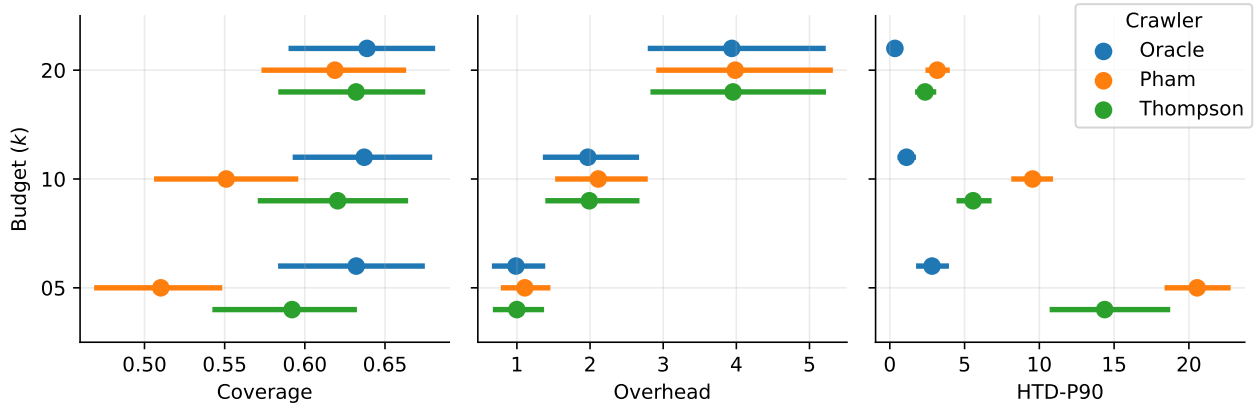

Figure 2: Average coverage, overhead, and HTD-P90 for the three crawlers across all 53 domains.

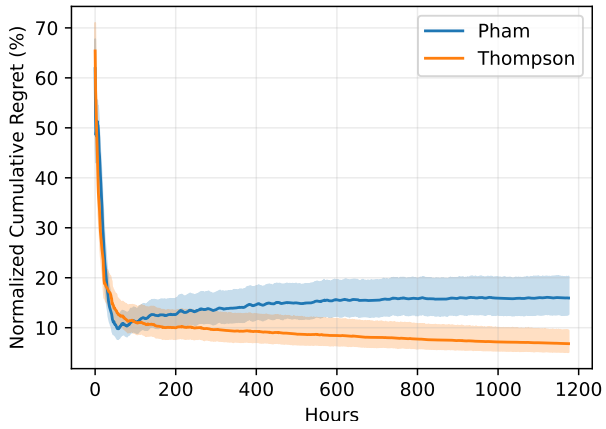

Figure 3: Cumulative regret averaged over domains.

experiments. It may be possible to tune these hyperparameters to improve performance, but we did not do so in our experiments.

3.3 Evaluating Convergence to Oracle

In the bandits literature, researchers commonly evaluate and compare algorithms by measuring *cumulative regret* over time [19, 22]. The cumulative reward $r(t)$ at time t is the sum of all rewards received up to, and including, time t. An algorithm's regret is the difference between the cumulative reward obtained by an optimal strategy $r^*(t)$ and that obtained by the algorithm's strategy $r(t)$. A strong algorithm that balances exploration and exploitation effectively will quickly converge to and maintain a low regret [19, 22].

We define the cumulative reward to be the total number of new pages discovered up to and including time t. Because the number of new pages varies across domains, we compute the *normalized regret* $100 \times \frac{r^*(t)-r(t)}{r^*(t)}$ to make the regret curves comparable across domains. Using a per-domain budget of $k = 5$, Figure 3 shows the average cumulative regret (and 95% bootstrapped confidence intervals) over all 53 domains for both the Pham and Thompson crawlers relative to the Oracle crawler; that is, we use the Oracle crawler's regret as the optimal regret $r^*(t)$.

We observe that both crawlers quickly converge to under 20% regret within the first few days. The Pham crawler reaches around 10% regret in approximately 75 hours, but bounces back to approximately 15% cumulative regret over time and stabilizes there. The Thompson crawler, on the other hand, smoothly progresses over time and converges to approximately 6% cumulative regret by the end of the study period. We conclude that the Thompson crawler balances exploration and exploitation more effectively than the Pham crawler.

3.4 Evaluating Crawl Performance

The regret curves in Figure 3 demonstrate that the Thompson crawler effectively balances exploration and exploitation, which helps it to efficiently converge to a strategy that is 6% worse than the oracle on average in terms of cumulative number of pages discovered. There are, however, other aspects of a content discovery crawler's performance that these curves do not reflect. In this section, we also evaluate the efficiency of the Pham and Thompson crawlers (measured using *overhead* as described in Section 2.1) and the timeliness of the crawlers' discoveries (measured using *HTD-P90* as described in Section 2.1).

We use the first week of data to allow each of the crawlers in our experiment to "warm up", and use the remaining six weeks to evaluate their performance in their steady state. After each run of a crawler on a given domain and with a given budget, we compute coverage, overhead, and HTD-P90 as defined in Section 2.1. To summarize a crawler's performance with a given budget, we average across domains and compute 95% confidence intervals for the average using 1,000 bootstrap samples.

Figure 2 shows the average coverage, overhead, and HTD-P90 (and 95% confidence intervals) for each crawler at each budget k. Starting with the left-most panel, we see that the oracle's coverage is just under 65% on average and is not sensitive to the budget (i.e. it remains at 65% for all budgets). This suggests that a strong refresh policy may "saturate" at relatively small budgets, supporting the widely adopted assumption in the literature that most new content can be discovered from a relatively small number of source pages. The fact that the Oracle crawler's coverage plateaus at 65% also

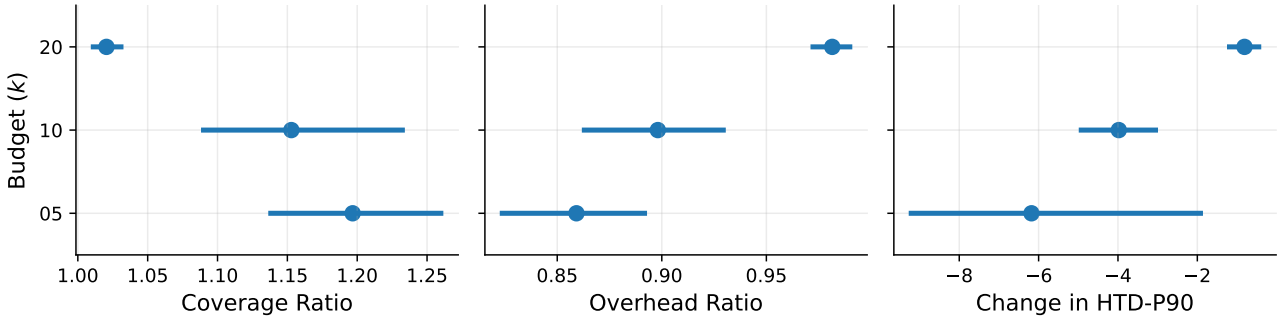

Figure 4: Average change in coverage, overhead, and HTD-P90 of Thompson crawler relative to Pham crawler over 53 domains.

shows that the "long tail" of new content may not be efficiently discoverable (i.e. discoverable without refreshing all known sources). This data is consistent with Dasgupta et al. [10], who observed that 25% of new content may not be efficiently discoverable. The Pham and Thompson crawlers both approach this plateau given sufficiently large budget, but the Thompson crawler is competitive with the Oracle crawler even at the lowest budget $k = 5$.

In the center panel of Figure 2, we show the overhead of each crawler at each budget (the number of source refreshes required to discover one new page). All crawlers become less efficient as the budget is increased, again supporting the assumption that the majority of new content is discoverable from a relatively small number of sources. In general, the Oracle crawler is most efficient and the Pham crawler is least efficient on average, but there is considerable overlap in the confidence intervals.

Finally, in the right-most panel of Figure 2, we show the 90th percentile of the distribution over hours to discovery of target pages. The Oracle crawler discovers most pages within 3-4 hours of when they first appear, even at the lowest budget. The Pham and Thompson crawler are much more sensitive to budget; they discover most pages in under 5 hours when $k = 20$, but can take much longer to discover pages when $k = 5$ (20 hours on average for Pham, and 15 hours for Thompson).

We draw several conclusions from Figure 2. First, as Dasgupta et al. [10] observed, a significant portion of new pages simply cannot be discovered efficiently (i.e. without refreshing all known sources). A comprehensive discovery crawler may need to plan occasional "full refreshes" in order to cover the long tail. Second, there is a strong relationship between budget and efficiency for all crawlers; even the Oracle crawler becomes less efficient as the budget is increased. We conjecture that this may be due to the fact that a large budget is not needed at every time step. For instance, we may be wasting most of the budgeted refreshes in the middle of the night when most pages are not changing very much. A discovery crawler that adaptively adjusts the budget based on expected yield over time may be able to remain efficient while still enjoying the improvements in coverage and HTD-P90 observed at high budgets.

3.4.1 Comparing Thompson to Pham. We now directly compare the Pham and Thompson crawlers. To compare the two crawlers, we designate the Thompson crawler as the *comparison* and Pham as the *reference*. For each metric x, budget k, and domain d, we compute x_{kd}^{comp} and x_{kd}^{ref}. To compare coverage and efficiency, we use the ratio $x_{kd}^{\text{comp}}/x_{kd}^{\text{ref}}$ and compute the average ratio across domains. To compare HTD-P90, we use the difference $x_{kd}^{\text{comp}} - x_{kd}^{\text{ref}}$ and compute the average difference across domains. Figure 4 shows the average ratio/difference for each budget along with the 95% confidence intervals (computed using 1,000 bootstrap samples). Overall, we observe statistically significant improvements over the Pham crawler across all metrics and all budgets (but the gap narrows considerably as the budget becomes larger). At the smallest budget (where a careful refresh policy is most important), we see that, on average, the Thompson crawler discovers 20% more pages, finds pages 6 hours earlier, and requires 14 fewer refreshes per 100 discoveries to do so (i.e. we see a relative reduction in overhead by 14%).

4 CONCLUSION

Quickly and efficiently discovering new pages on the web is important for a wide variety of applications. Pham et al. [17] showed that its possible to reduce the number of fetches required to run content discovery crawls by proposing an algorithm that sidesteps the need for "snapshot crawls" in the method introduced by Dasgupta et al. [10]. In this paper, we identified a modeling decision in the Pham crawler that leads to a sub-optimal exploration/exploitation trade-off, which impacts the effectiveness of a discovery crawl. We proposed an alternative, the Thompson crawler, that addresses this issue. On a collection of 4,070 source pages from 53 news domains, we showed that, on average, the Thompson crawler discovers 20% more new pages, finds pages 6 hours earlier, and requires 14 fewer refreshes per 100 pages discovered than the Pham crawler.

Thompson sampling is a flexible approach to designing algorithms that can effectively balance exploration and exploitation in a wide variety of problems. Our work is the first to show that Thompson sampling is an effective technique for guiding discovery crawls. Thompson sampling can accommodate a wide variety of objectives (e.g. we might incorporate clickstream data into the reward as in [14]). We can also design probabilistic models that reflect rich domain-specific structure. For instance, placing structured priors on the expected yield λ that model dependencies between hours that are close in time may help to improve the rate at which crawlers can learn the yield model parameters. While flexible, Thompson sampling depends on Bayesian inference, which can be prohibitively expensive to do at "web scale" and "web speed". This challenge presents new opportunities to push the state of the art in Bayesian inference for problems on the web.

REFERENCES

[1] Eytan Adar, Jaime Teevan, Susan T Dumais, and Jonathan L Elsas. 2009. The web changes everything: understanding the dynamics of web content. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*. 282–291.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2 (2002), 235–256.

[3] Michael K Bergman. 2001. The deep web: surfacing hidden value. *Journal of electronic publishing* 7, 1 (2001).

[4] Brian E Brewington and George Cybenko. 2000. How dynamic is the Web? *Computer Networks* 33, 1-6 (2000), 257–276.

[5] Maria Carla Calzarossa and Daniele Tessera. 2015. Modeling and predicting temporal patterns of web content changes. *Journal of Network and Computer Applications* 56 (2015), 115–123.

[6] Olivier Chapelle and Lihong Li. 2011. An empirical evaluation of Thompson sampling. *Advances in neural information processing systems* 24 (2011).

[7] Junghoo Cho and Hector Garcia-Molina. 2003. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems (TODS)* 28, 4 (2003), 390–426.

[8] Junghoo Cho and Alexandros Ntoulas. 2002. Effective change detection using sampling. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 514–525.

[9] Edward G Coffman Jr, Zhen Liu, and Richard R Weber. 1998. Optimal robot scheduling for web search engines. *Journal of scheduling* 1, 1 (1998), 15–29.

[10] Anirban Dasgupta, Arpita Ghosh, Ravi Kumar, Christopher Olston, Sandeep Pandey, and Andrew Tomkins. 2007. The discoverability of the web. In *Proceedings of the 16th ACM International World Wide Web Conference*. 421–430.

[11] Kanik Gupta, Vishal Mittal, Bazir Bishnoi, Siddharth Maheshwari, and Dhaval Patel. 2016. AcT: Accuracy-aware crawling techniques for cloud-crawler. *World Wide Web* 19, 1 (2016), 69–88.

[12] Vassiliki Hatzi, B Barla Cambazoglu, and Iordanis Koutsopoulos. 2016. Optimal web page download scheduling policies for Green Web crawling. *IEEE Journal on Selected Areas in Communications* 34, 5 (2016), 1378–1388.

[13] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric J Horvitz. 2019. Staying up to date with online content changes using reinforcement learning for scheduling. *Advances in Neural Information Processing Systems* 32 (2019).

[14] Damien Lefortier, Liudmila Ostroumova, Egor Samosvat, and Pavel Serdyukov. 2013. Timely crawling of high-quality ephemeral new content. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 745–750.

[15] Christopher Olston and Marc Najork. 2010. Web crawling. *Foundations and Trends® in Information Retrieval* 4, 3 (2010), 175–246.

[16] Christopher Olston and Sandeep Pandey. 2008. Recrawl scheduling based on information longevity. In *Proceedings of the 17th international conference on World Wide Web*. 437–446.

[17] Kien Pham, Aécio Santos, and Juliana Freire. 2018. Learning to Discover Domain-Specific Web Content. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*. 432–440.

[18] Kira Radinsky and Paul N Bennett. 2013. Predicting content change on the web. In *Proceedings of the sixth ACM international conference on Web search and data mining*. 415–424.

[19] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. 2018. A tutorial on Thompson sampling. *Foundations and Trends® in Machine Learning* 11, 1 (2018), 1–96.

[20] Uri Schonfeld and Narayanan Shivakumar. 2009. Sitemaps: above and beyond the crawl of duty. In *Proceedings of the 18th international conference on World wide web*. 991–1000.

[21] Matt Southern. 2022. Google Considers Reducing Webpage Crawl Rate. https://www.searchenginejournal.com/google-crawl-rate/434265/. Accessed: 2022-02-03.

[22] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[23] William R Thompson. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* 25, 3-4 (1933), 285–294.

[24] William R Thompson. 1935. On the theory of apportionment. *American Journal of Mathematics* 57, 2 (1935), 450–456.