

# Adversarial Bandits Policy for Crawling Commercial Web Content

Shuguang Han<sup>1</sup>, Michael Bendersky<sup>1</sup>, Przemek Gajda<sup>2</sup>, Sergey Novikov<sup>2</sup>, Marc Najork<sup>1</sup>, Bernhard Brodowsky<sup>2</sup> and Alexandrin Popescul<sup>3\*</sup>

1. Google Research, 1600 Amphitheatre Parkway, Mountain View, CA, USA

2. Google Zurich, Brandschenkestrasse 110, 8002 Zurich, Switzerland

3. Pinterest, 651 Brannan St, San Francisco, CA, USA

hanshuguang,bemike,pgajda,sergeyn,najork,bernhardb@google.com,apopescul@pinterest.com

## ABSTRACT

The rapid growth of commercial web content has driven the development of shopping search services to help users find product offers. Due to the dynamic nature of commercial content, an effective recrawl policy is a key component in a shopping search service; it ensures that users have access to the up-to-date product details. Most of the existing strategies either relied on simple heuristics, or overlooked the resource budgets. To address this, Azar et al. [5] recently proposed an optimization strategy *LambdaCrawl* aiming to maximize content freshness within a given resource budget. In this paper, we demonstrate that the effectiveness of *LambdaCrawl* is governed in large part by how well future content change rate can be estimated. By adopting the state-of-the-art deep learning models for change rate prediction, we obtain a substantial increase of content freshness over the common *LambdaCrawl* implementation with change rate estimated from the past history. Moreover, we demonstrate that while *LambdaCrawl* is a significant advancement upon existing recrawl strategies, it can be further improved upon by a unified multi-strategy recrawl policy. To this end, we adopt the *K*-armed adversarial bandits algorithm that can provably optimize the overall freshness by combining multiple strategies. Empirical results over a large-scale production dataset confirm its superiority to *LambdaCrawl*, especially under tight resource budgets.

## CCS CONCEPTS

• **Information systems** → **Web crawling**; *E-commerce infrastructure*; *Search engine architectures and scalability*.

## KEYWORDS

Predictive Crawling, Commercial Web Crawling, Adversarial Bandit

### ACM Reference Format:

Shuguang Han, Michael Bendersky, Przemek Gajda, Sergey Novikov, Marc Najork, Bernhard Brodowsky and Alexandrin Popescul. 2020. Adversarial Bandits Policy for Crawling Commercial Web Content. In *Proceedings of The Web Conference 2020 (WWW '20)*, April 20–24, 2020, Taipei, Taiwan. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3366423.3380125>

\*Work done when Alexandrin Popescul was at Google.

This paper is published under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 International (CC-BY-NC-ND 4.0) license. Authors reserve their rights to disseminate the work on their personal and corporate Web sites with the appropriate attribution.

WWW '20, April 20–24, 2020, Taipei, Taiwan

© 2020 IW3C2 (International World Wide Web Conference Committee), published under Creative Commons CC-BY-NC-ND 4.0 License.

ACM ISBN 978-1-4503-7023-3/20/04.

<https://doi.org/10.1145/3366423.3380125>

## 1 INTRODUCTION

Over recent years, there has been a substantial growth of commercial web content such as product listings, business websites and online shopping platforms. To better access such information, many search engines including Google and Bing have developed shopping search services for commerce-related query intents [3, 48]. Providing up-to-date information in product search results is a challenging task, in large part due to the dynamic nature of the web content in general [2, 9, 12], and commercial content in particular [26].

In this paper, we specifically focus on commercial web pages that sell products, and refer to such pages as the *offer* pages. Since price correctness is one of the most important concerns for shopping search engines [26], our goal is to optimally synchronize the price information stored in our local database with the true price.<sup>1</sup>

Simple synchronization strategies such as using the merchant-provided feed are insufficient, since most merchants do not have the capacity to provide low-latency feeds and sufficiently monitor their quality [26]. Thus, a periodic crawling of offer pages remains a vital component for production-grade commercial content search engines. Scheduling frequent recrawls for all pages is impractical due to the massive amount of commercial offers. Therefore, production systems often need to select a subset of offers for crawling.

Cho and Garcia-Molina [16] examined a simple page selection heuristic which chose web pages based on their change rates. Surprisingly, this strategy even under-performed the uniform heuristic that recrawled every page at an equal probability. This was mainly because such a proportional strategy overspent resources on frequently-changed pages, for which, no matter how often they were recrawled, the content might change at the next time period. As a result, the best strategy should penalize such web pages.

A recent study by Azar et al. [5] went beyond the above simple heuristics, and proposed a constrained optimization algorithm named *LambdaCrawl* (to be consistent with the nomenclature used by Kolobov et al. [29]) to seek for the optimal recrawl rates under resource constraints. The resulting best strategy aligned with our expectation on suppressing crawls for highly-dynamic pages.

Despite the recent advancements, there remain several limitations of existing studies. First of all, previous work mainly concentrated on recrawling generic web content, whereas little is known about the recrawling for commercial web pages. As shown by Han et al. [26], there are several major differences (e.g., the content change dynamics) between these two types of web pages. Thus, the first thing we examine in this paper – using production service

<sup>1</sup>While we focus on price correctness optimization, we believe that the proposed approach also generalizes to other attributes like product availability.

crawling data – is the adaptability of existing recrawl strategies to the domain of commercial web content crawling.

In addition, most existing recrawl strategies assumed the knowledge of change rate beforehand, which is usually unavailable in practice. To obtain such information, prior studies have developed various estimation approaches [24, 36, 38]. The simplest one is to estimate it from the past history [17, 21, 25, 39]. However, this approach suffers from the cold start issue and is subject to the feedback loop, which motivated follow-up studies [26, 40, 44] to incorporate predictive features that are universally available or relatively static, e.g. page content, when predicting change rate. In this paper, we follow such an approach and further experiment the utility of employing deep learning models for change rate prediction.

Another limitation of existing studies is that they all focused on developing a *single strategy*, and assumed that by applying such a strategy, one can achieve the optimal freshness. We argue (based on empirical observation) that such a goal is hard to achieve in reality. A single strategy often focuses on crawling one page type, either because of the nature of the strategy or due to the inherent parameter estimation bias; therefore, it may fail to effectively crawl other page types. Besides, in view of the dynamic change patterns for commercial content [26], it is possible that a strategy performs the best at one time, while it is sub-optimal at another time.

To address the above limitation, we examine the applicability of developing a unified, multi-strategy approach with reinforcement learning, in which we adopt the  $K$ -armed adversarial bandits algorithm [4] and treat each of the existing strategies as an arm. It starts with a random selection from multiple (potentially sub-optimal) strategies and gradually learn the best combination of strategies (i.e. *policy*) over time. The reward mechanism in bandit algorithms can guide the optimization towards the maximal content freshness, even if this goal is not present in each underlying strategy.

To summarize, the main contributions of our work are as follows:

- We are the first to provide an extensive survey and analysis of existing recrawl strategies for commercial web content. In particular, we conduct an evaluation of *LambdaCrawl* [5], and empirically demonstrate its superiority over the heuristic-based strategies for crawling commercial web content.
- We demonstrate that parameter estimation substantially affects the effectiveness of a recrawl strategy. Our proposed predictive model that takes both the past history and the metadata information into account improves upon the history-based models by a very large margin.
- We propose a  $K$ -armed adversarial bandits approach that combines multiple recrawl strategies under a unified policy with provable freshness guarantees. Our experiments show that this approach achieves a higher freshness than any single strategy, and is robust even under very tight resource budgets. In particular, such an approach outperforms the state-of-the-art *LambdaCrawl* strategy, even if *LambdaCrawl* is not included as a candidate strategy.
- Finally, we examine the contributions of individual strategies to the adversarial bandits approach, and discover the importance of exploration to achieve the optimal freshness. We empirically show that when the uniform random sampling is included, adversarial bandits learn an overall better policy.

## 2 RELATED WORK

Developing an appropriate content recrawling strategy is a long-standing research problem for web crawlers. A large body of work has been done since the 1990s in the context of maximizing freshness for crawling systems [13, 18, 36]. The core of this problem lies in the fact that web content changes dynamically [2, 9, 12, 14, 24]. Accordingly, many of the previous studies have been devoted to estimating the content change frequency [15, 17, 19, 45].

There are in general two types of estimation approaches: history-based approach and model-based approach. The former estimates content change based on the past change history [14, 23, 32]. Cho and Garcia-Molina [17] discovered that this can be inaccurate particularly when the change history is partially observable. Thus, they proposed a few improved estimators for better approximating the change rate. Recently, studies have begun to treat this estimation task as an online learning problem, and leveraged the reinforcement learning algorithms. Kolobov et al. [28] proposed a *LambdaLearnAndCrawl* approach to jointly estimate content change rate and optimize content freshness. Upadhyay et al. [46] applied the Explore-Then-Commit algorithm that iteratively estimated change rate and applied it for freshness-driven crawling. Despite the recent advancements, change rate prediction remains a challenging task especially when web pages have limited history information.

The model-based approach assumes regularities behind content change so that the change rate can be estimated using predictive models. Indeed, through analyzing page updates for news websites, Calzarossa and Tessera [11, 12] discovered that the content change can be characterized by well-defined temporal patterns. Han et al. [26] also identified several temporal and content patterns for commercial content changes. This motivated researchers to incorporate temporal and content features when building machine learning models for change estimation. Radinsky and Bennett [40] hypothesized that web pages with similar content share similar change patterns, and therefore built a predictive model combining both the historical changes and the page content. Tan and Mitra [44] clustered web pages into groups based on content similarity, and then focused on crawling those groups with high change rates. Han et al. [26] tackled a similar problem for commercial content crawling. They found that metadata information such as product brand and merchant information are important indicators of content change. Therefore, those features are adopted for predicting content changes. Comparing to the history-based approach, these models relieve the cold start issue since page content and metadata are often easily accessible and are relatively static.

However, just having an accurate estimation for content change is insufficient. Another key component is establishing the update policy – the ways to apply the estimated change rate for production crawlers. Cho and Garcia-Molina [16] discovered that the strategy of crawling web pages proportionally to their change rate even underperformed the uniform random strategy. Later studies confirmed this finding, and found that the optimal strategy should penalize highly dynamic pages [17, 20]. Designing an optimal strategy for a production crawler is more sophisticated, largely due to the need for incorporating practical constraints such as resource limit and politeness restriction (to avoid overloading the same web host) [6]. Azar et al. [5] proposed a constraint optimization framework named

*LambdaCrawl*, which sought to find optimal crawling rates under resource limits. Kolobov et al. [29] extended this framework by incorporating the politeness constraint. One issue with these two studies is that the parameters (change rate and click rate) were assumed to be *known* beforehand, which is unrealistic in practice. As a result, Kolobov et al. [28] and Upadhyay et al. [46] further explored the reinforcement learning approaches to jointly learn parameters and optimize scheduling process.

A few other studies have also considered controlling crawl costs and incorporating constraints, but were solved in different ways. Eckstein et al. [22] injected the politeness constraint at server-side when scheduling recrawls. Olston and Pandey [39] distinguished web pages by content longevity, and only focused on crawling the persistent content, which led to fresher content with lower cost. Lefortier et al. [30], on the other hand, studied the crawling of ephemeral content. By adopting the resource allocation theory [27], Wolf et al. [47] proposed a two-stage content refresh policy, with the first stage determining the best crawling frequencies, and the second stage creating achievable crawling schedules.

Overall, existing studies did conduct extensive research on both content change prediction and crawler scheduling optimization. However, there remain several missing pieces. First, none of them carried out a thorough analysis of existing recrawl strategies, particularly for the recently proposed ones such as *LambdaCrawl* [5] and in the domain of commercial web content crawling. Second, it is unknown how parameter estimation would affect the effectiveness of different strategies. In this paper, we try to fill these gaps.

Moreover, one may notice that reinforcement learning algorithms have been adopted to develop better recrawl strategies [28, 46]. However, their major goals remain to be the development of one single strategy. We thus propose a  $K$ -armed adversarial bandit policy, aiming to integrate multiple strategies into a unified policy. In contrast to prior studies, this approach allows exploration, alleviates the systemic errors made by each individual strategy, and makes use of predictive models for content change rate.

It is worth noting that reinforcement learning has also been applied to focused crawlers [33, 41]. A focused crawler collects web pages regarding one topic through properly managing the hyperlink exploration process. Here, reinforcement learning was applied to model the rewarding of on-topic crawling for hyperlink exploration. However, our task has no hyperlink exploration aspect; it aims to optimally recrawl known pages, thus focusing on modeling clicks and updates, and combining them within a single framework.

### 3 METHODOLOGY

Before delving into the details for each recrawl strategy, we first provide a formal description of the problem. Suppose that we have a total number of  $n$  offers ( $o_1, o_2, \dots, o_n$ ) in our production system. We can represent each offer  $o_i$  as a time series, with the data point at time step  $t$  denoted by a vector of three attributes ( $\mu_i^t, v_i^t, \Delta_i^t$ ).

<sup>2</sup> Here,  $\mu_i^t \in \mathbb{R}^+$  represents the click rate,  $v_i^t \in \mathbb{R}^+$  refers to the impression rate and  $\Delta_i^t \in [0, 1]$  indicates the probability of price change. Because of the price change, we need to periodically recrawl

offers so that our local database can store the latest information. We then define a recrawl rate  $\rho_i^t \in \mathbb{R}^+$  to represent the amount of recrawls we make for offer  $o_i$  at time step  $t$ .

Furthermore, we denote offer  $o_i$ 's latest price at time step  $t$  as  $r_i^t$ , which might or might not be observed by a recrawl strategy. In the meantime, each strategy also maintains a price  $l_i^t$  in local database for serving end users. At access time, users will see the right price only if  $r_i^t$  matches  $l_i^t$ , i.e.  $\mathbb{1}(r_i^t = l_i^t)$ , where  $\mathbb{1}(\cdot)$  is a binary indicator. The price match is a function of two factors – the price change history and the recrawl history.

Production crawlers also need to deal with resource constraints. Here, we assume a fixed crawling budget of  $b$  offers at each time step. Our goal is to find recrawling rates  $\mathcal{P}^t = (\rho_1^t, \rho_2^t, \dots, \rho_n^t)$  that can maximize the overall utility given the resource restrictions. For web crawling, freshness is often adopted to represent the utility. In this paper, we employ two widely-used freshness metrics [5, 16]: click-weighted freshness and page-level freshness. An offer is said to be fresh if its local price  $l_i^t$  matches its true price  $r_i^t$ . Here, since each page corresponds to one product offer, we refer to the page-level freshness as the offer-level freshness. The click-weighted freshness measures the percentage of clicks when users see the right price, whereas the offer-level freshness examines the proportion of offers with price information updated regardless of clicks.

For convenience, we summarize the notation in Table 1. We will provide a more detailed explanation, in the following sections, for the symbols that have not been introduced yet.

**Table 1: A list of notations we used in this paper.**

$o_i$	a product offer $i$	$t$	a time step
$n$	total number of offers	$b$	offer crawling budget at $t$
$\mu_i^t$	click rate of $o_i$ at $t$	$v_i^t$	impression rate of $o_i$ at $t$
$l_i^t$	local price of $o_i$ at $t$	$r_i^t$	true price of $o_i$ at $t$
$\rho_i^t$	recrawl rate of $o_i$ at $t$	$\Delta_i^t$	change rate of $o_i$ at $t$
$x_{i,k}^t$	the reward of applying a recrawl strategy $k$ for offer $i$ at $t$		
$x_k^t$	the accumulated reward by aggregating $x_{i,k}^t$ for all offers		
$A_t$	the recrawling strategy we applied at $t$		

#### 3.1 Existing Recrawl Strategies

Cho and Garcia-Molina [15] summarized two recrawl heuristics: the uniform strategy and the proportional strategy. In terms of crawling product offers, the uniform strategy crawls every offer at an equal rate, whereas the proportional strategy crawls offers relatively to a certain attribute. [15] only studied one proportional strategy based on the estimated change rate. In this paper, we extend it to other attributes including the estimated click rate and impression rate, both could help improve the click-weighted freshness.

Specifically, we study the following three proportional strategies: the change-weighted strategy, the click-weighted strategy and the impression-weighted strategy, whose recrawling rates are provided in Table 2. The underlying assumption for the change-weighted strategy is that if all of the offers with price changes have been crawled, there will be no stale content. The click-weighted strategy

<sup>2</sup>Note that we apply the notation  $t$  both to the time step itself (a time point) and the *time interval* ( $t - 1, t$ ). Therefore, some notations may refer to the time interval, while other refers to the time point. The exact definitions should be clear from the context.

assumes that if all of the clicked offers have been crawled before the click, the click-weighted freshness will be optimal. The impression-weighted strategy shares a very similar presumption.

**Table 2: Existing recrawl strategies and their crawling rates. Here,  $\sum \rho_i^t = b$  with  $b$  denoting the per time step budget.**

Recrawl strategy	Recrawl rate $\rho_i^t$
Uniform	$b \cdot 1/n$
Change weighted ( $\Delta$ )	$b \cdot \Delta_i / \sum \Delta_i$
Click weighted ( $\mu$ )	$b \cdot \mu_i / \sum \mu_i$
Impression weighted ( $v$ )	$b \cdot v_i / \sum v_i$
LambdaCrawl ( $\lambda$ )	$\frac{\sqrt{\frac{\mu_i \Delta_i}{\lambda} - \Delta_i}}{1 - \Delta_i}, \quad \lambda = \left( \frac{\sum \frac{\sqrt{\mu_i \Delta_i}}{1 - \Delta_i}}{b + \sum \frac{\Delta_i}{1 - \Delta_i}} \right)^2$

Azar et al. [5] went beyond the heuristic strategies, and proposed a theoretically optimal recrawl strategy named *LambdaCrawl* which solved the recrawl task as a constrained optimization problem [7]. The core of this problem was to represent the price match function  $\mathbb{1}(l_i^t = r_i^t)$ . With independence and consistency assumptions for  $\mu_i$ ,  $\Delta_i$  and  $\rho_i$ , the authors derived an analytical form for this function. By applying the Lagrange multipliers, they obtained a closed-form solution for the optimal recrawling rates, as shown in Table 2. *LambdaCrawl* favors the offers with greater click and change rates, but also penalizes the ones with too frequent changes.

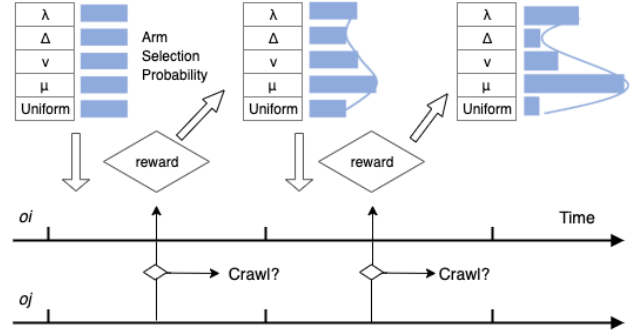
The theoretical optimality for *LambdaCrawl* is contingent on the constancy of content change rate, which does not hold empirically. Prices are known to change more frequently during holiday and promotional seasons. Even excluding the seasonality fluctuation, prices also exhibit weekly and hourly patterns [26]. Besides, the optimal recrawling rate in *LambdaCrawl* is parameterized by  $\mu_i$  and  $\Delta_i$ , both are unknown beforehand and will contain prediction errors during estimation. This motivates us to seek an alternative approach that is less sensitive to the constancy assumption, and can potentially relieve the parameter estimation errors.

## 3.2 K-armed Adversarial Bandits Crawl Policy

**3.2.1 Overview.** In this section, we model the recrawling task as a  $K$ -armed adversarial bandits (KAB) problem [4], where each strategy from Table 2 is treated as an arm. At each time step, we pick one arm based on its historical performance, determine offers to crawl using the picked arm<sup>3</sup>, observe rewards and update the arm’s performance. By repeating this process, we are able to improve the arm selection process as time goes on. This can be illustrated by Figure 1, in which we start with choosing every arm equally at time step 1, and through aggregating rewards, the click-weighted strategy and *LambdaCrawl* are getting progressively more preference as time passes. Since *Arm* and *Reward* are the two important concepts in this paper, we list their definitions below.

<sup>3</sup>At each time step, we loop through all of the offers, compute the recrawling rate based on Table 2, and finally, select a subset of them for crawling.

- **Arm:** any recrawl strategy listed in Table 2
- **Reward:** the increase of click-weighted freshness (§3.2.2)



**Figure 1: An illustration of the adversarial bandits crawl policy with five arms: uniform,  $\mu$ ,  $v$ ,  $\Delta$  and  $\lambda$ , as shown in Table 2. The blue bars denote the actual arm selection probabilities (at step 1, 100, 1000) from our later experiments.**

Compared to adopting a single strategy, using adversarial bandits is more advantageous in the following aspects: (1) incorporating multiple strategies allows us to explore offers from different angles, making it more robust to the errors made by an individual strategy; (2) different from stochastic bandit algorithms, adversarial bandits do not make stationarity assumptions on reward distribution [10], which is a better choice because the reward we employed – click-weighted freshness – is dynamic. An alternative to adversarial bandits is to assign a fixed amount of resources for each strategy and then run all of them simultaneously. This is suboptimal since different strategies often have overlaps. For example, offers selected by the impression-weighted strategy are significantly overlapped with the click-weighted strategy. As shown in Figure 1, with a high selection probability for the click-weighted strategy, our approach suppresses the choice of impression-weighted strategy, even though they can achieve similar performance if being applied individually (see Table 5). Besides, the optimal resource assignment might change over time because of the seasonality and temporal dynamics for clicks and price changes [26].

**3.2.2 Formalization.** We formalize the adversarial bandits approach as follows: assume that we have  $K$  candidate arms, and let  $x_k^t \in [0, 1]$  indicate the reward one will receive by adopting the  $k$ -th arm at time step  $t$ . The goal is to choose a sequence of arms  $(A_1, A_2, \dots, A_t, \dots)$  so that by applying those arms, the **regret** ( $R$ ) of not using the best arm at every time step is minimized, as shown in Formula (1). Note that instead of sampling an arm per offer per time step, we only select one arm at each time step and apply it across all offers. This avoids a joint optimization of resource budget over multiple strategies since the crawling rates for each time step have already incorporated the resource constraint.

$$R = \left( \max_{k \in [K]} \sum_t x_k^t \right) - \mathbb{E} \left[ \sum_t x_{A_t}^t \right] \quad (1)$$

The reward  $x_k^t$  (or  $x_{A_t}^t$ ) is calculated by accumulating per offer utility  $x_{i,k}^t$  – the payoff by applying the  $k$ -th arm for offer  $o_i$  at time

step  $t$ . It is defined in the following way: if crawling an offer helps update the local price to the latest, we believe such a crawl is useful, and thus assign a positive reward. As shown in Formula (2), we use the local price from time step  $(t - 1)$  to verify that it will not match the true price at time step  $t$ ; if these prices do match, there will be no utility gain for crawling this offer. To align with the click-weighted freshness, we boost the reward by click rate. In fact, this utility measures the *increase* of click-weighted freshness between two time steps. The recrawling rate  $\rho_i^t$  is used to denote whether the offer will be crawled because there will be no utility gain if not crawling the offer. In addition, we include a normalization term to rescale the reward to  $[0, 1]$ .

$$x_k^t = \sum_i x_{i,k}^t = \sum_i \left( \frac{1}{\sum_i \mu_i^t \rho_i^t} \cdot \mu_i^t \cdot \rho_i^t \cdot \mathbb{1}(l_i^{t-1} \neq r_i^t) \right) \quad (2)$$

Minimizing regret  $R$  is equivalent to maximizing the expected reward, the second term in Formula (1), since the accumulated reward for applying the best strategy at every time step is a constant factor. Furthermore, based on the definition in Formula (2), the time aggregated reward actually represents the click-weighted freshness, meaning that our proposed adversarial bandits approach is essentially optimizing the click-weighted freshness.

**3.2.3 Implementation.** We adopt the standard EXP3 algorithm for implementing the KAB policy as it provides a nearly optimal regret bound [4].<sup>4</sup> Algorithm 1 provides the implementation details.

This algorithm starts by initializing a uniform importance weight  $w_k^0=1$  for each arm (line 1). Here,  $k$  indicates the  $k$ -th arm and 0 stands for the time step  $t=0$ . At each time step  $t$ , we compute a probability distribution  $Q^t$  with each element  $q_k^t$  denoting the probability of choosing the  $k$ -th arm (line 3).  $q_k^t$  is determined by the importance weight  $w_k^t$  and the exploration probability  $\gamma$ . Next, we sample an arm  $A_t$  from  $Q^t$  (line 4), and compute the corresponding offer crawling rate  $\rho_i^t$  based on Table 2 (line 7). Note that the resource budget  $b$  has already been integrated into  $\rho_i^t$ . While crawling, we also aggregate the reward  $x_{A_t}^t$  for the sampled arm (line 9). At last, we update  $w_k^t$  for the arm  $A_t$  based on the reward (line 11), and then repeat.

This algorithm solves the exploitation-exploration trade-off in reinforcement learning by introducing the exploration probability  $\gamma$ . It samples arms proportionally to their past performances at a probability of  $1 - \gamma$ , and also maintains a probability of  $\gamma$  to choose a random arm for exploration. When reporting the model performance, we set  $\gamma = 0.1$  as it produces the best performance.

When applying Algorithm 1, we denote each time step as two hours. This significantly reduces the computation overhead as we only need to loop through all offers 12 times per day, making it a more practical solution. Moreover, it provides sufficient time for the algorithm to accumulate meaningful reward and historical performance statistics. As for implementation, we scale the per time step crawling rate to bi-hourly rate. More specifically, the crawling rate in Table 2 is computed for each time step, we need to multiply it by (12 hours / time unit) for the bi-hourly rate.

<sup>4</sup>We also experimented with several recent adversarial bandits algorithms such as EXP3-IX [37] and EXP3++ [43], but they produced similar results to EXP3, which we thus adopt in the remainder of the paper due to its conceptual simplicity.

---

### Algorithm 1 The $K$ -armed adversarial bandits approach

---

**Parameter:**  $\gamma \in [0, 1]$

- 1:  $\forall k$ , set  $w_k^0 = 1$
- 2: **for** time  $t = 1, 2, \dots, T$  **do**
- 3:  $\forall k$ , set  $q_k^t = (1 - \gamma) \frac{w_k^t}{\sum_{j=1}^K w_j^t} + \frac{\gamma}{K}$
- 4: Sample an arm  $A_t \sim Q^t : (q_1^t \dots q_K^t)$
- 5: Set the reward  $x_{A_t}^t = 0$
- 6: **for** offer  $i = 1, 2, \dots, n$  **do**
- 7:     Compute  $o_i$ 's crawling rate  $\rho_i^t$  for arm  $A_t$  (Table 2)
- 8:     Schedule  $\rho_i^t$  crawls\* and update local price  $l_i^t$
- 9:     Update reward  $x_{A_t} \ += x_{i,A_t}^t$
- 10: **end for**
- 11:  $\forall k$   $w_k^{t+1} = w_k^t \cdot \exp(\frac{\gamma}{K} \cdot \mathbb{1}(k = A_t) \cdot \frac{x_{A_t}^t}{q_i^t})$
- 12: **end for**

\*: During simulation, this means to select from the crawling log.

---

### 3.3 Parameter Estimation

Deploying the above recrawl strategies (except the uniform crawl) requires knowing the click rate, the impression rate and the change rate. The simplest way is to estimate them from past history [17]. Due to the cold start and feedback loop issues, researchers have exploited predictive models utilizing page content and metadata information for parameter estimation [26, 40]. We follow the predictive modeling approach, and employ both **metadata** and past **history** information for better parameter estimation accuracy.

Specifically, we model the price change prediction as a classification task, for which we want to predict whether an offer's price will change in the next day. Similarly, for click and impression prediction, we forecast whether an offer will be clicked or impressed in the next day. The prediction outputs will be directly used as  $\mu$ ,  $\nu$  and  $\Delta$  when computing the crawling rate. Here, we set our prediction horizon at the daily granularity since click and impression statistics are aggregated on the daily basis.

We adopt two change history features, including the monthly price change frequency and the most recent change, because of their best performance in [26]. We also include a set of click and impression history features, which are strong signals for predicting future clicks and impressions. All of the features and their descriptions are provided in Table 3. The product category information comes from Google Shopping product taxonomy<sup>5</sup>. The change frequencies, click and impression statistics are treated as numerical dense features, and the metadata information is modeled by sparse features and is embedded into a low-dimensional space [35].

For each prediction task, we train three models – one using the **metadata** features, one using the **history** features and a third with **metadata + history** features. For each model, we adopt the feed-forward deep neural network (DNN) model with TensorFlow DNNClassifier [1], where we set three hidden layers to 256, 128 and 64 hidden units in each layer. We use ReLU (Rectified Linear Unit)

<sup>5</sup>See here: <https://www.google.com/basepages/producttype/taxonomy.en-US.txt>

**Table 3: History and metadata features used in our models.**

History features for the predictive model	
Change frequency (1 month)	Price change frequency in last month
Most recent change	Time since the most recent change
Clicks (1 day)	Clicks at yesterday
Clicks (1 week)	Daily clicks in the past week
Clicks (2 weeks)	Daily clicks in the past 2 weeks
Clicks (1 month)	Daily clicks in the past month
Impressions (1 day)	Impressions at yesterday
Impressions (1 week)	Daily impressions in the past week
Impressions (2 weeks)	Daily impressions in the past 2 weeks
Impressions (1 month)	Daily impressions in the past month
Metadata features for the predictive model	
Brand	Unique ID for a brand
Condition	Condition: new, used or refurbished
Country	Country code
Day of Week	Day of week for the prediction time
Language	Offer page language
Merchant	Unique ID for a merchant
Product category	Product category

as the activation function for hidden units, and choose the Adagrad algorithm to optimize the cross-entropy loss. To deal with overfitting, we adopt both L1 and L2 regularization and set both to 0.001. Note that we also experimented multiple sets of hyper-parameters, the evaluation results remain to be similar.

## 4 DATA AND EVALUATION

In this paper, we sampled 1.3 million offers indexed by Google Shopping search engine and scheduled hourly crawls for these offers. This helped us acquire a full observation of price history. The samples came from two types: (a) random *uniform samples* from the entire corpus of offers; and (b) *click weighted samples*, to better represent popular offers with clicks. In total, we crawled billions of offer page snapshots from 2018/08/01 to 2019/04/10. Note that not all of the page snapshots can be successfully downloaded because the crawling requests might be rejected.

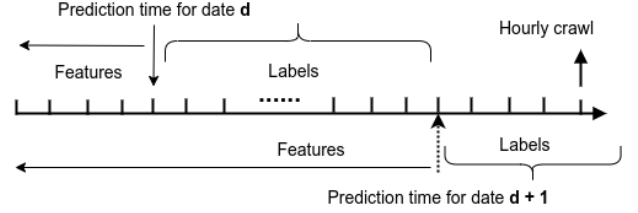
Besides the hourly crawls, we also have access to the click and impression information. Such information is used for building predictive models, defining evaluation metrics and examining the performance of recrawl strategies. Since clicks and impressions are aggregated on a daily basis, our predictive models and recrawl strategies are also tested at the granularity of a day.

### 4.1 Evaluating Predictive Models

**4.1.1 Dataset.** To build a predictive model, we need a set of training examples, validation examples and testing examples. The validation and testing examples are extracted from the uniform samples to simulate the production needs. The training examples are derived from both the uniform samples and click-weighted samples to reduce the imbalance of click and impression labels.

As shown in Figure 2, each example is created in the following way. For each simulated prediction date  $d$ , we define the prediction time  $t$  as the beginning of  $d$  (12:00 am). Features are then extracted

from the crawling logs up to time  $t$ . Hourly crawls after  $t$  provide a full observation for the future price information, which helps us generate a binary *label* reflecting whether the price will change in the next day. Similarly, the click and impression information on  $d+1$  are used to create a binary click/impression label denoting whether the offer will be clicked/impressed in the next day. By shifting the prediction date  $d$  and repeating the above process, we create a set of training, testing and validation examples.



**Figure 2: Dataset generation process for the prediction dates  $d$  and  $d + 1$ . A vertical line denotes a hourly crawl.**

The training, validation and testing datasets are created with data from different dates. Particularly, data from 2018/08/01 to 2018/12/31 is used for training, data from 2019/01/01 to 2019/01/09 is used for validation and the rest is used for testing. In total, we obtain 0.6 million validation examples, 8 million testing examples and 100 million training examples. In the testing and validation data, the positive/negative label ratios are 1:20 for price change, 1:75 for click and 1:6 for impression, whereas for the training data, we observe higher positive/negative ratios due to the involvement of click-weighted samples. The ratios become 1:20 for price change, 1:1 for click and 4:1 for impression. Note that since the uniform samples are picked randomly from the entire corpus, many are obsolete, removed or have no price extracted. This causes the number of testing and validation examples being lower than expected.

**4.1.2 Evaluation Metric.** To evaluate our predictive models, we adopt the AUC metric (Area Under Receiver Operating Characteristic Curve). A value of 0.5 means a random guess while 1.0 indicates a perfect prediction. The evaluation is conducted on the testing dataset while the validation dataset is used for model selection.

### 4.2 Evaluating Recrawl Strategies

The ultimate goal of building the predictive models is to provide better parameters (i.e.,  $\mu$ ,  $\nu$  and  $\Delta$ ) for different recrawl strategies. To further understand the effectiveness of each strategy, we follow the below evaluation process.

**4.2.1 Evaluation Process.** We design the evaluation process by simulating our production system. Specifically, when examining a recrawl strategy  $k$ , we maintain a local price  $l_i^t(k)$  for each offer  $o_i$ . This price will be updated if  $o_i$  is crawled by the strategy  $k$ . Meanwhile, the hourly crawls from our dataset provide a full observation for the price history which can then be used to infer the ground-truth price  $r_i^t$ . By comparing  $l_i^t(k)$  and  $r_i^t$ , we can compute the freshness of  $o_i$  at any simulated prediction time.

This process can be illustrated by Figure 3. To avoid reusing the training data, we exclude the crawling data before 2019/01/10

at the evaluation time. In addition, we set up a *warm-up stage* from 2019/01/10 to 2019/03/10. During this time, we only apply a recrawl strategy but do not report its performance. This allows sufficient time for the adversarial bandits policy to accumulate historical performances for each candidate arm. Finally, we report the freshness at the *evaluation stage* from 2019/03/11 to 2019/04/10.



**Figure 3: The evaluation process for a recrawl strategy.**

**4.2.2 Evaluation Metric.** As mentioned in Section 3, when evaluating a recrawl strategy, we adopt both click-weighted freshness and offer-level freshness, and compute them at the daily granularity. Specifically, on a particular date  $d$ , the offer-level freshness  $of_d^k$  for a strategy  $k$  is defined as the percentage of offers that have price information updated at midnight ( $t_m=12:00am$ ), as shown in Formula (3). The click-weighted freshness  $cf_d^k$  is measured by the proportion of clicks for which the users see the right price, which is computed by Formula (4).  $\mu_i^t$  denotes the click rate at time step  $t$ .

$$of_k^d = \frac{1}{n} \sum_i \mathbb{1}(l_i^m(k) = r_i^m) \quad (3)$$

$$cf_k^d = \frac{1}{\sum_{t \in d} \sum_i \mu_i^t} \sum_{t \in d} \sum_i \mu_i^t \cdot \mathbb{1}(l_i^t(k) = r_i^t) \quad (4)$$

## 5 EXPERIMENTS

This section starts with evaluating the predictive models for parameter estimation. With the best performed models, we then report how well the predicted parameters can help promote different recrawl strategies. At last, we conduct an extensive assessment for our proposed K-armed adversarial bandits recrawling policy.

### 5.1 Predictive Modeling Performance

For each prediction task (price change prediction, click prediction and impression prediction), we compare three predictive models: one only using the metadata features, one only using the history features and the third using both features. Table 4 provides the testing AUCs for the three models. Those values are obtained in the following way – at first, we select the best model based on the validation dataset; then, we split our testing data into 20 subsets and apply the models on each subset; finally, we compute the average and standard deviation of AUCs over the 20 subsets.

Overall, the predictive models with only the metadata features achieved AUCs above 0.73, indicating the value of such features in all three prediction tasks. Particularly, for the price change prediction, it significantly outperforms the history features, and is only 2% shy of the best model. As for the click and impression predictions, using history features outperforms the metadata features, and a combination of both does not provide too much added value.

Click/impression prediction is an important research topic which has been the central task for many online services [34]. Given that the main focus of our paper is to build a better recrawl strategy, further prediction performance improvements are out of scope of this work. Instead, we simply apply the model with both metadata and history features due to its best performance across all tasks, as well as its ability of dealing with the cold-start issue.

**Table 4: Testing AUCs (and standard deviation) for predictive models. Numbers in bold (italic) denote that the results are significant comparing to the history (metadata) features.**

Task \ Model	Metadata	History	Metadata + History
Price change	0.860 (0.008)	<i>0.833 (0.011)</i>	<b>0.882 (0.007)</b>
Click	0.796 (0.021)	<i>0.948 (0.006)</i>	<i>0.949 (0.006)</i>
Impression	0.736 (0.008)	<i>0.896 (0.003)</i>	<i>0.895 (0.003)</i>

### 5.2 Heuristic Recrawl Strategies

This section examines four heuristic recrawl strategies from Table 2, including the uniform strategy, the change-weighted strategy, the click-weighted strategy and the impression-weighted strategy. To measure the performance of each strategy, we adopt both click-weighted freshness and offer-level freshness. As mentioned in Section 4.2.2, our goal is to maximize the **click-weighted freshness**, and potentially improve the offer-level freshness, if possible.

**5.2.1 Experiment Setup.** Following Figure 3, when evaluating a recrawl strategy, we apply it on both warm-up stage (01/10 to 03/10) and evaluation stage (03/11 to 04/10), but only report results in the second stage. Specifically, for each day in the evaluation stage, we compute one freshness value, which results in 31 values for 31 days. Meanwhile, since the crawl decisions are made stochastically, we repeat the whole evaluation process 100 times to reduce the randomness. In total, we obtain 3,100 values (100 runs  $\times$  31 days). Hereafter, we report the median of those values since they are not normally distributed. For the same reason, we adopt the Wilcoxon signed-rank test to examine statistical significance.

As shown in Table 2, the recrawl rate for each strategy contains several parameters: resource budget  $b$  and click/impression/change rate  $\mu_i/v_i/\Delta_i$ . During evaluation, we vary the relative percentage of resource budget  $b/n$  from 10% to 90%, with 10% as the step width. The rest of parameters, including  $\mu_i$ ,  $v_i$  and  $\Delta_i$ , are estimated either from the past history or using the predictive models. The following sections provide a more detailed evaluation for different recrawl strategies under the two parameter estimation methods.

**5.2.2 Parameter Estimation from Historical Frequency.** We firstly evaluate the four heuristic strategies with model parameters estimated from the historical frequencies (except the uniform policy). Such an estimation was applied and proven to be effective in many prior studies [17, 26, 40]. We denote them as *uniform*,  $\mu_h$ ,  $v_h$  and  $\Delta_h$ . Here, model parameters in  $\mu_h$ ,  $v_h$  and  $\Delta_h$  are estimated using the average daily price changes, clicks and impressions in the past 30 days. We choose 30 days because of its good performance and less sparsity than only using data from one day or one week.

**Table 5: Click-weighted freshness (median) for different recrawl strategies, and across various resource budgets. Numbers in bold denote the best strategy in each group; numbers with \* indicate the overall best strategy. † means KAB5 performs significantly better (p-value < 0.05) than  $\lambda_{ml}$ , and the corresponding numbers are the median of increase percentages.**

Group	Policy \ Budget	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform	Uniform	0.7460	0.8162	0.8512	0.8720	0.8840	0.8928	0.9029	0.9105	0.9165
History	$\Delta_h$	0.1951	0.2508	0.2817	0.3007	0.3171	0.3294	0.3412	0.3507	0.3549
	$\mu_h$	0.8840	0.8933	0.8978	0.9006	0.9030	0.9051	0.9068	0.9083	0.9097
	$\nu_h$	<b>0.9013</b>	<b>0.9153</b>	<b>0.9224</b>	<b>0.9266</b>	<b>0.9295</b>	0.9314	0.9328	0.9341	0.9351
	$\lambda_h = \lambda_{\mu_h, \Delta_h}$	0.8843	0.9079	0.9184	0.9244	0.9291	<b>0.9322</b>	<b>0.9350</b>	<b>0.9371</b>	<b>0.9385</b>
Predictive Models	$\Delta_{ml}$	0.4067	0.5177	0.5857	0.6339	0.6664	0.6935	0.7159	0.7344	0.7483
	$\mu_{ml}$	0.8976	0.9098	0.9199	0.9251	0.9357	0.9380	0.9437	0.9463	0.9494
	$\nu_{ml}$	0.8835	0.9054	0.9222	0.9301	0.9320	0.9362	0.9409	0.9435	0.9461
	$\lambda_{ml} = \lambda_{\mu_{ml}, \Delta_{ml}}$	<b>0.8983</b>	<b>0.9152</b>	<b>0.9284</b>	<b>0.9384</b>	<b>0.9424</b>	<b>0.9456</b>	<b>0.9486</b>	<b>0.9505</b>	<b>0.9518</b>
Adversarial Bandit	KAB5 +% † (vs. $\lambda_{ml}$ )	<b>0.9027*</b> +0.93%†	<b>0.9238*</b> +0.71%†	<b>0.9373*</b> +0.44%†	<b>0.9425*</b> +0.35%†	<b>0.9478*</b> +0.33%†	<b>0.9497*</b> +0.28%†	<b>0.9517*</b> +0.23%†	<b>0.9533*</b> +0.22%†	<b>0.9543*</b> +0.15%†

**Table 6: Offer-level freshness (median) for different recrawl strategies, and across various resource budgets. Numbers in bold denote the best strategy in each group; numbers with \* indicate the overall best strategy. † means KAB5 performs significantly better (p-value < 0.05) than  $\lambda_{ml}$ , and the corresponding numbers are the median of increase percentages.**

Group	Policy \ Budget	10%	20%	30%	40%	50%	60%	70%	80%	90%
Uniform	Uniform	0.7426*	0.8135*	0.8454*	0.8637*	0.8758*	0.8849*	0.8917*	0.8971*	0.9015*
History	$\Delta_h$	0.1589	0.2031	0.2268	0.2423	0.2539	0.2632	0.2713	0.2782	0.2846
	$\mu_h$	0.0724	0.1008	0.1269	0.1513	0.1746	0.1966	0.2175	0.2375	0.2565
	$\nu_h$	0.1239	0.1582	0.1792	0.1946	0.2063	0.2157	0.2237	0.2304	0.2363
	$\lambda_h = \lambda_{\mu_h, \Delta_h}$	<b>0.3424</b>	<b>0.4884</b>	<b>0.5722</b>	<b>0.6247</b>	<b>0.6630</b>	<b>0.6921</b>	<b>0.7130</b>	<b>0.7291</b>	<b>0.7427</b>
Predictive Models	$\Delta_{ml}$	0.4611	0.5779	0.6404	0.6812	0.7106	0.7333	0.7513	0.7662	0.7786
	$\mu_{ml}$	0.4447	0.5894	0.6608	0.7033	0.7314	0.7528	0.7693	0.7830	0.7943
	$\nu_{ml}$	0.6033	0.7095	0.7584	0.7875	0.8082	0.8233	0.8351	0.8446	0.8523
	$\lambda_{ml} = \lambda_{\mu_{ml}, \Delta_{ml}}$	<b>0.6550</b>	<b>0.7561</b>	<b>0.8023</b>	<b>0.8289</b>	<b>0.8463</b>	<b>0.8588</b>	<b>0.8684</b>	<b>0.8759</b>	<b>0.8818</b>
Adversarial Bandit	KAB5 +% † (vs. $\lambda_{ml}$ )	<b>0.6604</b> +1.07%†	<b>0.7635</b> +1.09%†	<b>0.8064</b> +0.72%†	<b>0.8319</b> +0.49%†	<b>0.8495</b> +0.44%†	<b>0.8612</b> +0.36%†	<b>0.8709</b> +0.38%†	<b>0.8782</b> +0.34%†	<b>0.8845</b> +0.35%†

Table 5 presents the evaluation results for the click-weighted freshness. Same as Cho and Garcia-Molina [15], we find that crawling offers proportionally to their change rates ( $\Delta_h$ ) performs the worst. The main reason is that  $\Delta_h$  spends too much resource on the highly dynamic content which might change again at the time users access them. We also see that  $\mu_h$  and  $\nu_h$  outperform the uniform strategy. This is because the click-weighted freshness weighs more on the clicked offers. In addition, we observe that  $\nu_h$  performs better than  $\mu_h$ . One potential reason is that the impression information is less sparse than the click information, making it a better estimator for future clicks. Indeed, the percentage of positive impression label (with at least one impression in the next day) is around 15% whereas the percentage is only 1.5% for click.

Table 6 reports the offer-level freshness. Despite being the secondary metric, we still want to maintain it at a certain level. Here,

the *uniform* strategy outperforms all others, which is due to the following reasons. First, most of the above strategies are designed to optimize the click-weighted freshness rather than the offer-level freshness. Second, daily statistics from the past 30 days are relatively static. Using such statistics may end up choosing similar offers, while the non-selected offers will never be updated. This remains true if the parameters are estimated from predictive models since the metadata features are also relatively static. On the contrary, the uniform strategy brings diverse offers each time. Besides, we see that  $\mu_h$  and  $\nu_h$  perform less well than  $\Delta_h$ . This is expected because  $\mu_h$  and  $\nu_h$  only predict future clicks, whereas such information is not considered in the offer-level freshness.

**5.2.3 Parameter Estimation from Predictive Models.** Here, we evaluate the same four heuristic strategies except the parameters are estimated from the predictive models. We thus rename them as  $\mu_{ml}$ ,



$v_{ml}$  and  $\Delta_{ml}$ . The results are provided in Table 5 and Table 6. Overall, comparing to the history-based parameter estimation, the predictive models bring substantial lifts for both freshness metrics. For the click-weighted freshness,  $\Delta_{ml}$  doubles the performance of  $\Delta_h$ . For the offer-level freshness,  $\Delta_{ml}$ ,  $\mu_{ml}$  and  $v_{ml}$  are improved as much as four to five times. One reason is that the predictive models smooth the crawling rate so that the highly changed/clicked offers will not have dramatic differences with the rarely changed/clicked offers. Besides, the predictive models also provide meaningful predictions for those offers with limited or no past histories.

### 5.3 LambdaCrawl

Different from the heuristic strategies, *LambdaCrawl* determines its recrawling rate by combining both click rate and change rate. Again, we consider two *LambdaCrawl* strategies based on the parameter estimation approaches —  $\lambda_h$  that combines  $\mu_h$  and  $\Delta_h$ , and  $\lambda_{ml}$  that integrates  $\mu_{ml}$  and  $\Delta_{ml}$ . In terms of the implementation, we directly apply Algorithm 1 from Azar et al. [5], where we set the goal as to optimize the click-weighted freshness. The evaluation results are also presented in Table 5 and Table 6.

According to Table 5, we find that  $\lambda_h$  does effectively integrate  $\mu_h$  and  $\Delta_h$ , producing a better click-weighted freshness than the strategies that apply them separately. With more accurate parameters from the predictive models,  $\lambda_{ml}$  brings a further improvement for the click-weighted freshness. In addition, despite we have set the goal to optimize the click-weighted freshness, *LambdaCrawl* also achieves surprisingly high offer-level freshness (see Table 6). This might be due to that there are too few clicks (less than 1.5%) in the evaluation dataset; therefore, optimizing the click-weighted freshness also optimizes the offer-level freshness implicitly.

In addition, observing that the impression-weighted strategy also performs very well on the click-weighted freshness, we further experimented with replacing the click rate with impression rate in *LambdaCrawl*. Specifically, we tested two new strategies  $\lambda_{v_h, \Delta_h}$  and  $\lambda_{v_{ml}, \Delta_{ml}}$ . However, both of them performed less well than  $\lambda_{\mu_{ml}, \Delta_{ml}}$ ; thus, we did not report the results in this paper.

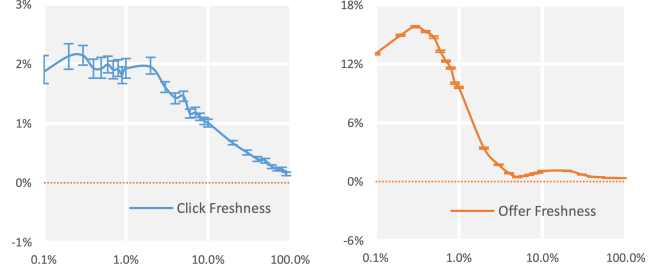
### 5.4 K-armed Adversarial Bandits Policy

**5.4.1 Overall Performance.** We firstly examine a KAB policy with the following strategies:  $\{\text{uniform}, \Delta_{ml}, \mu_{ml}, v_{ml}, \lambda_{ml}\}$ . We name it as KAB5 since it contains five candidate strategies. Our expectation is that through combining different strategies, it can achieve a better performance than using each of them separately. Evaluation results for this policy are also provided in Table 5 and Table 6, which clearly show the effectiveness of KAB5 on boosting both the click-weighted freshness and offer-level freshness over the baselines. Besides, the performance improvement is consistent across the nine resource budgets, indicating the robustness of the KAB approach.

**5.4.2 Varying Resource Budgets.** The above experiments only consider resource budgets from 10% to 90%. For production systems that crawl millions of offers, they often deal with a much tighter crawling budget. To understand the robustness of KAB5 over different resource budgets, we run a set of additional experiments with tighter resources ranging from 1% to 9% (1% as the step size) and 0.1% to 0.9% (0.1% as the step size). Then, we compute and plot the

freshness increases over  $\lambda_{ml}$  in Figure 4. Since the data is not normally distributed, we plot the medians, and the confidence intervals are computed for medians [8]. Here, we use  $\lambda_{ml}$  for benchmark as it is the best baseline for the click-weighted freshness.

The freshness change curves are above zero across all resource budgets, meaning that KAB5 consistently outperforms  $\lambda_{ml}$  on both click-weighted and offer-level freshness metrics. The improvements are especially pronounced for very tight (< 1%) budgets. All of the improvements are significant at p-value < 0.05 level.



**Figure 4: Freshness change (median, 95% CI) over  $\lambda_{ml}$  for KAB5 across different resource budgets. The horizontal axis denotes budgets (log scale) and the vertical axis indicates freshness increase/decrease (positive/negative value).**

**5.4.3 Arm Selection.** Since  $\lambda_{ml}$  is one of the arms in KAB5, it is possible that  $\lambda_{ml}$  will always be selected given its superior performance. To better understand how KAB5 chooses arms, we plot the arm selection probability (line 3 in Algorithm 1) at each time step in Figure 5. Here, we only plot the probabilities up to the time step 1,000 as we only have three months of data for evaluation (see Figure 3), and the probabilities are updated every two hours.

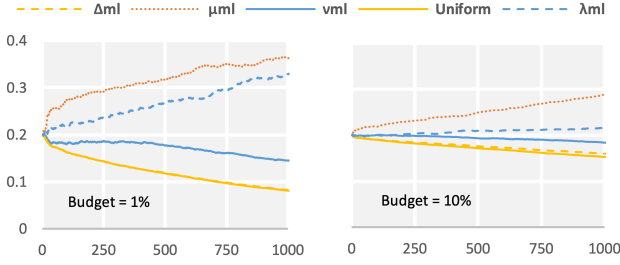
According to Figure 5, both  $\lambda_{ml}$  and  $\mu_{ml}$  receive probabilities higher than 0.2, the initialized uniform probability, indicating the utility of the two strategies. However,  $\mu_{ml}$  is shown to be more preferable than  $\lambda_{ml}$ , which differs from Table 5 — in case of being applied individually,  $\lambda_{ml}$  is better. One potential reason is that offers selected by  $\mu_{ml}$  are less diverse so that certain offers might never be selected if we apply such a strategy exclusively, whereas, in KAB5, those offers can be handled by other strategies. Besides, although there are several strategies receiving low selection probabilities, they also provide significant merits, particularly the uniform strategy. We will discuss this in more detail in the next section. Also, with more resources available, the arm selection probabilities become less imbalanced as there is more overlap across strategies.

**5.4.4 KAB without LambdaCrawl.** Considering the extra computation cost for *LambdaCrawl*, this section examines the possibility of excluding  $\lambda_{ml}$  from KAB. Specifically, we evaluate the following KAB policies. Here, we use KABX to denote a policy with X arms.

- KAB2 with candidates  $\{\mu_{ml}, \Delta_{ml}\}$
- KAB3<sup>6</sup> with candidates  $\{\mu_{ml}, \Delta_{ml}, \text{Uniform}\}$
- KAB4 with candidates  $\{\mu_{ml}, \Delta_{ml}, v_{ml}, \text{Uniform}\}$

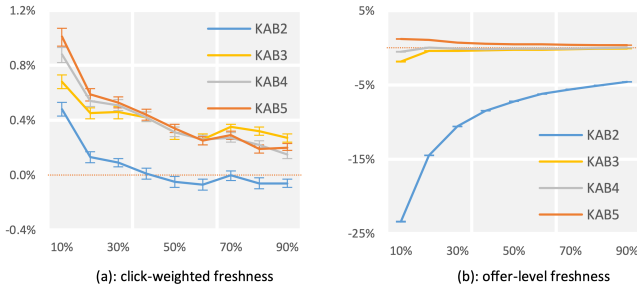
Figure 6 plots the freshness increase over  $\lambda_{ml}$  for the above three KAB policies and KAB5. As shown in Figure 6, the gap between

<sup>6</sup>We also tried a KAB3 with  $\{\mu_{ml}, \Delta_{ml}, v_{ml}\}$ , which performed similarly to KAB2.



**Figure 5: Arm selection probability for KAB5 under resource budgets 1% and 10%. The horizontal axis denotes time step.**

each KAB policy and  $\lambda_{ml}$  closes gradually (to zero) with the increase of resource budgets. This is expected because with more and more resources, the offers chosen (to be crawled) by different policies will have more overlaps. In the case of 100% resources, all offers are selected, making no differences among policies.



**Figure 6: Freshness change (median, 95% CI) over  $\lambda_{ml}$  for four KAB policies and across multiple resource budgets. The horizontal axis denotes budgets and the vertical axis indicates freshness increase/decrease (positive/negative value).**

Same as the  $\lambda_{ml}$ , KAB2 adopts both click rate and change rate. Although it falls behind  $\lambda_{ml}$  on the offer-level freshness, KAB2 achieves a comparable performance as  $\lambda_{ml}$  on the click-weighted freshness, particularly when the crawling budget exceeds 30%. On the one hand, the relatively good performances (both click-weighted and offer-level freshness) of  $\lambda_{ml}$  demonstrate its effectiveness on combining the click rate and the change rate, which cannot be easily achieved by other means. On the other hand, the on-par click-weighted freshness performance motivates us to further improve the KAB2 policy, especially for the offer-level freshness.

To boost the offer-level freshness, we include the uniform strategy as a new arm for the KAB policy because of its best offer-level freshness performance (Table 6). The new policy, named KAB3, does work well as expected. It significantly improves the offer-level freshness over KAB2, though it is still slightly lower than  $\lambda_{ml}$ . More importantly, KAB3 substantially lifts the click-weighted freshness, which outperforms  $\lambda_{ml}$  by a very large margin. This is due to the fact that the uniform strategy enables exploration by crawling hard-to-predict offers. Besides, a further integration of the  $v_{ml}$  strategy as in KAB4 provides an additional boost for the offer-level freshness, but the click-weighted freshness remains the same.

Finally, we integrate  $\lambda_{ml}$  in the KAB policy, thereby creating a KAB5 policy with five strategies. According to Figure 6, it not only produces a good click-weighted freshness but also further promotes

the offer-level freshness. KAB5 outperforms the  $\lambda_{ml}$  strategy on both freshness metrics. This is the best KAB policy we have and its results have been reported in Table 5 and Table 6.

To summarize, through an extensive experimentation of different KAB policies, we discover several advantages of the KAB approach. First, KAB can achieve reasonably good freshness even without  $\lambda_{ml}$ , suggesting that it is a legitimate alternative to *LambdaCrawl*. Moreover, the performance can be improved after including the *LambdaCrawl*. Second, adding the uniform strategy is critical for the KAB policy, which not only boosts the offer-level freshness but also promotes the click-weighted freshness. Third, the KAB policy provides a generic framework for integrating different recrawl strategies. Any strategy can be included or removed easily.

## 6 CONCLUSIONS AND FUTURE WORK

### 6.1 Conclusions

This paper studied the content recrawling problem in the context of building an effective production crawler for commercial content. In particular, we aimed to design a recrawl policy that not only maximizes content freshness but also considers resource limits.

We started by examining a number of existing recrawl strategies and discovered that the recently-proposed *LambdaCrawl* outperformed all other strategies, but its performance was dependent on an accurate estimation of future click rates and change rates. To this end, we proposed state-of-art deep neural models for parameter estimation (§3.3). These models not only produced the best prediction accuracy, but also resulted in a significant improvement of content freshness over the common *LambdaCrawl* implementation with parameters estimated from the past history.

To further improve upon the existing best practices, we proposed a  $K$ -armed adversarial bandits approach in §3.2, which treated each of the existing strategies as an arm and iteratively selected arms based on their historical performances. Empirical results from §5.4 demonstrated the superiority of this approach over *LambdaCrawl*, and its robustness across different resource budgets. Furthermore, the proposed approach also provided an effective framework for combining multiple recrawl strategies, which achieved a comparable performance as the best baseline model (*LambdaCrawl*) even without including it as a candidate strategy.

### 6.2 Future Work

The results in this paper suggest several future research directions. First, despite the fact that our experiments with two other recent bandit approaches, EXP3++ [43] and EXP3-IX [37], did not show much improvement beyond the standard EXP3, other bandit-based approaches may still be helpful. For example, instead of firstly selecting a recrawl strategy and then selecting offers to crawl, we can build a contextual bandit [31] to directly predict crawling rates based on the offer context.

Second, although we focus on commercial content crawling, we believe that our proposed approach can easily generalize to other crawling scenarios such as crawling news content and event information. In addition, adversarial bandits can be helpful in monitoring applications beyond crawling. E.g., with an increasing trend of publishing wireless sensor outputs on the Web, this can be an effective mechanism to update the latest sensor information for IoT [42].

## REFERENCES

- [1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: a system for large-scale machine learning. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*. 265–283.
- [2] 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 2nd ACM International Conference on Web Search and Data Mining*. 282–291.
- [3] Qingyao Ai, Yongfeng Zhang, Keping Bi, Xu Chen, and W Bruce Croft. 2017. Learning a hierarchical embedding model for personalized product search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 645–654.
- [4] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM J. Comput.* 32, 1 (2002), 48–77.
- [5] Yossi Azar, Eric Horvitz, Eyal Lubetzky, Yuval Peres, and Dafna Shahaf. 2018. Tractable near-optimal policies for crawling. *Proceedings of the National Academy of Sciences* 115, 32 (2018), 8099–8103.
- [6] Ricardo Baeza-Yates and Carlos Castillo. 2002. Balancing volume, quality and freshness in web crawling. In *Proceedings of the 2nd International Conference on Hybrid Intelligent Systems*. 565–572.
- [7] Dimitri P Bertsekas. 2014. *Constrained optimization and Lagrange multiplier methods*. Academic Press.
- [8] Martin Bland. 2015. *An introduction to medical statistics*. Oxford University Press (UK).
- [9] Brian E Brewington and George Cybenko. 2000. How dynamic is the web? *Computer Networks* 33, 1-6 (2000), 257–276.
- [10] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. 2012. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning* 5, 1 (2012), 1–122.
- [11] Maria Carla Calzarossa and Daniele Tessera. 2008. Characterization of the evolution of a news Web site. *Journal of Systems and Software* 81, 12 (2008), 2336–2344.
- [12] 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.
- [13] Carlos Castillo. 2005. Effective Web Crawling. *SIGIR Forum* 39, 1 (June 2005), 55–56.
- [14] Junghoo Cho and Hector Garcia-Molina. 2000. The Evolution of the Web and Implications for an Incremental Crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases*. Morgan Kaufmann Publishers Inc., 200–209.
- [15] Junghoo Cho and Hector Garcia-Molina. 2000. Synchronizing a database to improve freshness. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. 117–128.
- [16] Junghoo Cho and Hector Garcia-Molina. 2003. Effective page refresh policies for web crawlers. *ACM Transactions on Database Systems* 28, 4 (2003), 390–426.
- [17] Junghoo Cho and Hector Garcia-Molina. 2003. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)* 3, 3 (2003), 256–290.
- [18] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. 1998. Efficient crawling through URL ordering. *Computer Networks and ISDN Systems* 30, 1-7 (1998), 161–172.
- [19] Junghoo Cho and Alexandros Ntoulas. 2002. Effective change detection using sampling. In *Proceedings of the 28th International Conference on Very Large Data Bases*. 514–525.
- [20] 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.
- [21] Edith Cohen and Haim Kaplan. 2001. Refreshment policies for web content caches. In *IEEE INFOCOM 2001 - The Conference on Computer Communications - Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3. 1398–1406.
- [22] Jonathan Eckstein, Avigdor Gal, and Sarit Reiner. 2008. Monitoring an information source under a politeness constraint. *INFORMS Journal on Computing* 20, 1 (2008), 3–20.
- [23] Jenny Edwards, Kevin McCurley, and John Tomlin. 2001. An adaptive model for optimizing performance of an incremental web crawler. In *Proceedings of the 10th International World Wide Web Conference*. 106–113.
- [24] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet Wiener. 2003. A large-scale study of the evolution of web pages. In *Proceedings of the 12th International World Wide Web Conference*. 669–678.
- [25] Carrie Grimes, Daniel Ford, and Eric Tassone. 2008. Keeping a Search Engine Index Fresh: Risk and optimality in estimating refresh rates for web pages. *Proceedings of the 40th Symposium on the Interface: Computing Science and Statistics*, 1–14.
- [26] Shuguang Han, Bernhard Brodowsky, Przemek Gajda, Sergey Novikov, Mike Bendersky, Marc Najork, Robin Dua, and Alexandrin Popescu. 2019. Predictive Crawling for Commercial Web Content. In *The World Wide Web Conference*. 627–637.
- [27] Toshihide Ibaraki and Naoki Katoh. 1988. *Resource allocation problems: algorithmic approaches*. MIT press.
- [28] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric Horvitz. 2019. Staying up to Date with Online Content Changes Using Reinforcement Learning for Scheduling. In *Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36th International Conference on Machine Learning*.
- [29] 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*. 495–504.
- [30] 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.
- [31] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. 661–670.
- [32] Xiaoyong Li, Daren BH Cline, and Dmitri Loguinov. 2017. Temporal update dynamics under blind sampling. *IEEE/ACM Transactions on Networking (TON)* 25, 1 (2017), 363–376.
- [33] Andrew McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 1999. A machine learning approach to building domain-specific search engines. In *International Joint Conferences on Artificial Intelligence*, Vol. 99. Citeseer, 662–667.
- [34] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. 2013. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1222–1230.
- [35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).
- [36] Marc Najork and Allan Heydon. 2002. High-performance web crawling. In *Handbook of Massive Data Sets*. Springer, 25–45.
- [37] Gergely Neu. 2015. Explore no more: Improved high-probability regret bounds for non-stochastic bandits. In *Advances in Neural Information Processing Systems*. 3168–3176.
- [38] Christopher Olston and Marc Najork. 2010. Web crawling. *Foundations and Trends® in Information Retrieval* 4, 3 (2010), 175–246.
- [39] 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.
- [40] 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.
- [41] Jason Rennie, Andrew McCallum, et al. 1999. Using reinforcement learning to spider the web efficiently. In *International Conference on Machine Learning*, Vol. 99. 335–343.
- [42] Kay Romer, Benedikt Ostermaier, Friedemann Mattern, Michael Fahrmaier, and Wolfgang Kellerer. 2010. Real-time search for real-world entities: A survey. *Proc. IEEE* 98, 11 (2010), 1887–1902.
- [43] Yevgeny Seldin and Gábor Lugosi. 2017. An improved parametrization and analysis of the EXP3++ algorithm for stochastic and adversarial bandits. In *The 30th Annual Conference on Learning Theory*. Proceedings of Machine Learning Research, 1743–1759.
- [44] Qingzhao Tan and Prasenjit Mitra. 2010. Clustering-based incremental web crawling. *ACM Transactions on Information Systems (TOIS)* 28, 4 (2010), 17.
- [45] Qingzhao Tan, Ziming Zhuang, Prasenjit Mitra, and C Lee Giles. 2007. Efficiently detecting webpage updates using samples. In *International Conference on Web Engineering*. 285–300.
- [46] Utkarsh Upadhyay, Robert Busa-Fekete, Wojciech Kotłowski, David Pal, and Balazs Szorenyi. 2019. Learning to Crawl. *arXiv preprint arXiv:1905.12781* (2019).
- [47] Joel L Wolf, Mark S Squillante, PS Yu, Jay Sethuraman, and Leyla Ozsen. 2002. Optimal crawling strategies for web search engines. In *Proceedings of the 11th International Conference on World Wide Web*. 136–147.
- [48] Chao-Yuan Wu, Amr Ahmed, Gowtham Ramani Kumar, and Ritendra Datta. 2017. Predicting Latent Structured Intents from Shopping Queries. In *Proceedings of the 26th International Conference on World Wide Web*. 1133–1141.