Adversarial Bandits Policy for Crawling Commercial Web Content

Shuguang Han\textsuperscript{1}, Michael Bendersky\textsuperscript{1}, Przemek Gajda\textsuperscript{2}, Sergey Novikov\textsuperscript{2}, Marc Najork\textsuperscript{1}, Bernhard Brodowsky\textsuperscript{2} and Alexandrin Popescul\textsuperscript{3}.

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

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.\footnote{While we focus on price correctness optimization, we believe that the proposed approach also generalizes to other attributes like product availability.}

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 \textit{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...
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 (o1, o2, ..., on) in our production system. We can represent each offer oi as a time series, with the data point at time step t denoted by a vector of three attributes (μi, νi, Δi).

\[ \mu_i \in \mathbb{R}^+ \text{ click rate of } o_i \text{ at } t \]
\[ ν_i \in \mathbb{R}^+ \text{ impression rate of } o_i \text{ at } t \]
\[ Δ_i \in [0, 1] \text{ change rate of } o_i \text{ at } t \]

\[ x_{i,k}^t \text{ the reward of applying a recrawl strategy } k \text{ for offer } i \text{ at } t \]
\[ A_t \text{ the recrawling strategy we applied at } t \]

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

<table>
<thead>
<tr>
<th>( o_i )</th>
<th>a product offer i</th>
<th>t</th>
<th>a time step</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>total number of offers</td>
<td>b</td>
<td>offer crawling budget at t</td>
</tr>
<tr>
<td>( \mu_i )</td>
<td>click rate of ( o_i ) at t</td>
<td>( ν_i )</td>
<td>impression rate of ( o_i ) at t</td>
</tr>
<tr>
<td>( P_i^{t} )</td>
<td>local price of ( o_i ) at t</td>
<td>( r_i^{t} )</td>
<td>true price of ( o_i ) at t</td>
</tr>
<tr>
<td>( \rho_i^{t} )</td>
<td>recrawl rate of ( o_i ) at t</td>
<td>( \Delta_i^{t} )</td>
<td>change rate of ( o_i ) at t</td>
</tr>
<tr>
<td>( x_{i,k}^t )</td>
<td>the reward of applying a recrawl strategy ( k ) for offer ( i ) at ( t )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_{i}^t )</td>
<td>the accumulated reward by aggregating ( x_{i,k}^t ) for all offers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( A_t )</td>
<td>the recrawling strategy we applied at ( t )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 2: Existing recrawl strategies and their crawling rates. Here, \( \sum \rho_i^t = b \) with \( b \) denoting the per time step budget.

<table>
<thead>
<tr>
<th>Recrawl strategy</th>
<th>Recrawl rate ( \rho_i^t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>( b \cdot 1/n )</td>
</tr>
<tr>
<td>Change weighted (( \Delta ))</td>
<td>( b \cdot \Delta_i / \sum \Delta_i )</td>
</tr>
<tr>
<td>Click weighted (( \mu ))</td>
<td>( b \cdot \mu_i / \sum \mu_i )</td>
</tr>
<tr>
<td>Impression weighted (( \nu ))</td>
<td>( b \cdot \nu_i / \sum \nu_i )</td>
</tr>
<tr>
<td>LambdaCrawl (( \lambda ))</td>
<td>( \frac{\mu_i \Delta_i}{b + \sum \Delta_i} ), ( \lambda = \left( \frac{\sum \frac{\nu_i \Delta_i}{b + \sum \Delta_i}}{\Delta_i} \right)^2 )</td>
</tr>
</tbody>
</table>

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 \( K \) in this paper, we list their definitions below.

\[
\frac{\mu_i \Delta_i}{b + \sum \Delta_i}
\]

\[
\lambda = \left( \frac{\sum \frac{\nu_i \Delta_i}{b + \sum \Delta_i}}{\Delta_i} \right)^2
\]

\[\frac{\mu_i \Delta_i}{b + \sum \Delta_i} \]

\[\lambda = \left( \frac{\sum \frac{\nu_i \Delta_i}{b + \sum \Delta_i}}{\Delta_i} \right)^2 \]

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 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\(^5\), 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.

\[^5\text{At each time step, we loop through all of the offers, compute the recrawling rate based on Table 2, and finally, select a subset of them for crawling.}\]
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 will be no utility gain for crawling this offer. To align with the click-weighted freshness between two time steps. The recrawling rate $\rho_t^i$ 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 $[0, 1]$

$$x_t^i = \sum_{k} x_{t,k} = \sum_{k} \left( \frac{1}{\sum_{i} \mu_{t}^{(i)} \cdot \rho_{t}^{i} \cdot \Delta_{t}} \cdot (1_{t}^{i} \neq r_{t}^{i}) \right) \quad (2)$$

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].

Algorithm 1 provides the implementation details.

```
Algorithm 1 The $K$-armed adversarial bandits approach
Parameter: $\gamma \in [0, 1]$
1. $\forall k$, set $w_k^0 = 1$
2. for time $t = 1, 2, ..., T$
do
3. $\forall k$, set $q_k^t = (1 - \gamma) \cdot \frac{w_k^t}{\sum_{j} w_j^t} + \gamma$
4. Sample an arm $A_t \sim Q^t_t : (q_k^t)$
5. Set the reward $x_{t,A_t}^i = 0$
6. for offer $i = 1, 2, ..., n$
do
7. Compute $o_i$’s crawling rate $\rho_t^i$ for arm $A_t$ (Table 2)
8. Schedule $\rho_t^i$ crawls* and update local price $l_t^i$
9. Update reward $x_{A_t} = x_{A_t}^i$
end for
11. $\forall k$ $w_t^{k+1} = w_t^k \cdot \exp \left( \frac{v}{\gamma} \cdot (1 (k = A_t) x_{A_t}^k) \right)$
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 taxonomy5. 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)
Table 3: History and metadata features used in our models.

<table>
<thead>
<tr>
<th>History features for the predictive model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change frequency (1 month)</td>
</tr>
<tr>
<td>Most recent change</td>
</tr>
<tr>
<td>Clicks (1 day)</td>
</tr>
<tr>
<td>Clicks (1 week)</td>
</tr>
<tr>
<td>Clicks (2 weeks)</td>
</tr>
<tr>
<td>Clicks (1 month)</td>
</tr>
<tr>
<td>Impressions (1 day)</td>
</tr>
<tr>
<td>Impressions (1 week)</td>
</tr>
<tr>
<td>Impressions (2 weeks)</td>
</tr>
<tr>
<td>Impressions (1 month)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metadata features for the predictive model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brand</td>
</tr>
<tr>
<td>Condition</td>
</tr>
<tr>
<td>Country</td>
</tr>
<tr>
<td>Day of Week</td>
</tr>
<tr>
<td>Language</td>
</tr>
<tr>
<td>Merchant</td>
</tr>
<tr>
<td>Product category</td>
</tr>
</tbody>
</table>

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.](image)

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 \(l_i^k\) 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 \(r_i^k\). By comparing \(l_i^k\) and \(r_i^k\), 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 day $d$, the offer-level freshness of 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 $cf_k$ is measured by the proportion of clicks for which the users see the right price, which is computed by Formula (4). $p^t_i$ denotes the click rate at time step $t$.

$$of_k^d = \frac{1}{n} \sum_i \mathbb{1}(t^m_i(k) = r^m_i)$$  (3)

$$cf_k^d = \frac{1}{\sum_{t \in d} \sum_i p^t_i} \sum_{t \in d} \sum_i p^t_i \cdot \mathbb{1}(l^t_i(k) = r^t_i)$$  (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.

<table>
<thead>
<tr>
<th>Task \ Model</th>
<th>Metadata</th>
<th>History</th>
<th>Metadata + History</th>
</tr>
</thead>
<tbody>
<tr>
<td>Price change</td>
<td>0.860 (0.008)</td>
<td>0.833 (0.011)</td>
<td><strong>0.882 (0.007)</strong></td>
</tr>
<tr>
<td>Click</td>
<td>0.796 (0.021)</td>
<td>0.948 (0.006)</td>
<td>0.949 (0.006)</td>
</tr>
<tr>
<td>Impression</td>
<td>0.736 (0.008)</td>
<td>0.896 (0.003)</td>
<td>0.895 (0.003)</td>
</tr>
</tbody>
</table>

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 × 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 $\mu_i/\nu_i/\Delta_i$. 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 $\mu_i$, $\nu_i$ 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, $\mu_b$, $\nu_b$ and $\Delta_b$. Here, model parameters in $\mu_b$, $\nu_b$ and $\Delta_b$ 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.
Δworst. The main 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: 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.

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

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.

<table>
<thead>
<tr>
<th>Group</th>
<th>Policy \ Budget</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
<th>40%</th>
<th>50%</th>
<th>60%</th>
<th>70%</th>
<th>80%</th>
<th>90%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform</td>
<td>Uniform</td>
<td>0.7426</td>
<td>0.8135*</td>
<td>0.8454*</td>
<td>0.8637*</td>
<td>0.8758*</td>
<td>0.8849*</td>
<td>0.8917*</td>
<td>0.8971*</td>
<td>0.9015*</td>
</tr>
<tr>
<td>History</td>
<td>( \Delta_h )</td>
<td>0.1589</td>
<td>0.2031</td>
<td>0.2268</td>
<td>0.2423</td>
<td>0.2539</td>
<td>0.2632</td>
<td>0.2713</td>
<td>0.2782</td>
<td>0.2846</td>
</tr>
<tr>
<td></td>
<td>( \mu_h )</td>
<td>0.0724</td>
<td>0.1008</td>
<td>0.1269</td>
<td>0.1513</td>
<td>0.1746</td>
<td>0.1966</td>
<td>0.2175</td>
<td>0.2375</td>
<td>0.2565</td>
</tr>
<tr>
<td></td>
<td>( \nu_h )</td>
<td>0.1239</td>
<td>0.1582</td>
<td>0.1792</td>
<td>0.1946</td>
<td>0.2063</td>
<td>0.2157</td>
<td>0.2237</td>
<td>0.2304</td>
<td>0.2363</td>
</tr>
<tr>
<td></td>
<td>( \lambda_h = \lambda_{ml}\Delta_h )</td>
<td>0.3424</td>
<td>0.4884</td>
<td>0.5722</td>
<td>0.6247</td>
<td>0.6630</td>
<td>0.6921</td>
<td>0.7130</td>
<td>0.7291</td>
<td>0.7427</td>
</tr>
<tr>
<td>Predictive Models</td>
<td>( \Delta_{ml} )</td>
<td>0.4611</td>
<td>0.5779</td>
<td>0.6404</td>
<td>0.6812</td>
<td>0.7106</td>
<td>0.7333</td>
<td>0.7513</td>
<td>0.7662</td>
<td>0.7786</td>
</tr>
<tr>
<td></td>
<td>( \mu_{ml} )</td>
<td>0.4447</td>
<td>0.5894</td>
<td>0.6608</td>
<td>0.7033</td>
<td>0.7314</td>
<td>0.7528</td>
<td>0.7699</td>
<td>0.7830</td>
<td>0.7943</td>
</tr>
<tr>
<td></td>
<td>( \nu_{ml} )</td>
<td>0.6033</td>
<td>0.7095</td>
<td>0.7584</td>
<td>0.7875</td>
<td>0.8082</td>
<td>0.8233</td>
<td>0.8351</td>
<td>0.8446</td>
<td>0.8523</td>
</tr>
<tr>
<td></td>
<td>( \lambda_{ml} = \lambda_{ml}\Delta_{ml} )</td>
<td>0.6550</td>
<td>0.7561</td>
<td>0.8023</td>
<td>0.8289</td>
<td>0.8463</td>
<td>0.8588</td>
<td>0.8684</td>
<td>0.8759</td>
<td>0.8818</td>
</tr>
<tr>
<td>Adversarial Bandit</td>
<td>KAB5</td>
<td>0.6604</td>
<td>0.7635</td>
<td>0.8064</td>
<td>0.8319</td>
<td>0.8495</td>
<td>0.8612</td>
<td>0.8709</td>
<td>0.8782</td>
<td>0.8845</td>
</tr>
<tr>
<td></td>
<td>+% (vs. ( \lambda_{ml} ))</td>
<td>+1.07%†</td>
<td>+1.09%†</td>
<td>+0.72%†</td>
<td>+0.49%†</td>
<td>+0.44%†</td>
<td>+0.36%†</td>
<td>+0.38%†</td>
<td>+0.34%†</td>
<td>+0.35%†</td>
</tr>
</tbody>
</table>

Table 5 presents the evaluation results for the click-weighted freshness. Same as Cho and Garcia-Molina [15], we find that crawling offers disproportionately 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 \( \mu_{ml} \), \( \nu_{ml} \), \( \lambda_{ml} \), and \( \lambda_{ml}\Delta_{ml} \).
\(\nu_{\text{ml}}\) and \(\Delta_{\text{ml}}\). 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, \(\lambda_{\text{ml}}\) doubles the performance of \(\Delta_h\). For the offer-level freshness, \(\Lambda_{\text{ml}}, \mu_{\text{ml}}\) and \(\nu_{\text{ml}}\) 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 \(-\Delta_{\text{h}}\) and \(\lambda_{\text{h}}\), producing a better click-weighted freshness than the strategies that apply them separately. With more accurate parameters from the predictive models, \(\lambda_{\text{ml}}\) 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\%) to \(\lambda_{\text{h}}\) and \(\Delta_{\text{h}}\), and \(\lambda_{\text{ml}}\) that integrates \(\mu_{\text{ml}}\) and \(\Delta_{\text{ml}}\). 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 \(\lambda_{\text{h}}\) does effectively integrate \(\rho_{\text{h}}\) and \(\Delta_{\text{h}}\), producing a better click-weighted freshness than the strategies that use them separately. With more accurate parameters from the predictive models, \(\lambda_{\text{ml}}\) brings a further improvement for the click-weighted freshness. In addition, even though 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 evaluated the following strategies \(\lambda_{\text{ml}}, \Delta_{\text{ml}}\), and \(\nu_{\text{ml}}, \Delta_{\text{ml}}\). However, both of them performed less well than \(\lambda_{\text{ml}}, \Delta_{\text{ml}}\); 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: \{uniform, \(\Delta_{\text{ml}}, \mu_{\text{ml}}, \nu_{\text{ml}}, \Delta_{\text{ml}}\). We name it as KAB5 since it contains five candidate strategies. Our expectation is that 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 \(\lambda_{\text{ml}}\) 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 \(\lambda_{\text{ml}}\) 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 \(\lambda_{\text{ml}}\) 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.

5.4.3 Arm Selection. Since \(\lambda_{\text{ml}}\) is one of the arms in KAB5, it is possible that \(\lambda_{\text{ml}}\) 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 \(\lambda_{\text{ml}}\) and \(\mu_{\text{ml}}\) receive probabilities higher than 0.2, the initialized uniform probability, indicating the utility of the two strategies. However, \(\mu_{\text{ml}}\) is shown to be more preferable than \(\lambda_{\text{ml}}\), which differs from Table 5 — in case of being applied individually, \(\lambda_{\text{ml}}\) is better. One potential reason is that offers selected by \(\mu_{\text{ml}}\) 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 \(\lambda_{\text{ml}}\) from KAB. Specifically, we evaluate the following KAB policies. Here, we use KABX to denote a policy with \(X\) arms.

- KAB2 with candidates \{\(\mu_{\text{ml}}, \Delta_{\text{ml}}\)\}
- KAB5 with candidates \{\(\mu_{\text{ml}}, \Delta_{\text{ml}}, \text{Uniform}\)\}
- KAB4 with candidates \{\(\mu_{\text{ml}}, \Lambda_{\text{ml}}, \nu_{\text{ml}}, \text{Uniform}\)\}

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

---

6We also tried a KAB3 with \{\(\mu_{\text{ml}}, \Delta_{\text{ml}}, \nu_{\text{ml}}\)\}, which performed similarly to KAB2.
which outperforms weighted and offer-level freshness) of $\lambda$ will have more overlaps. In the case of 100% resources, all offers are selected, making no differences among policies.

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

Although it falls behind 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 the click-weighted 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].
Adversarial Bandits Policy for Crawling Commercial Web Content

REFERENCES


