Graph-Structured Crawling: Model and Approach

Mohammadhossein Bateni*
Google Research
New York, New York, USA
bateni@gmail.com

Lin Chen
Google Research
New York, New York, USA
linche@google.com

Hossein Esfandiari
Google Research
London, United Kingdom
esfandiari.hossein@gmail.com

Sasan Tavakkol
Google Research
New York, New York, USA
tavakkol@google.com

ABSTRACT

This study focuses on the crawling problem, i.e., keeping knowledge up to date where the information being studied may change over time. Examples of crawling include web crawling, where search engines aim to maintain a cache of the latest versions of web pages and, as a running example in this paper, detecting changes in the opening hours of parks on a map. Previous research on crawling often treat the objects as independent, despite all the connections and interactions they may have with one another. We propose using a graph structure to model the relationships between these objects and to better capture the influence of one object on another. We show that in the worst case, it is impossible to learn anything interesting in polynomial time, even though there exists a learnable structure that can be discovered in exponential time. This impossibility result motivates us to consider specific stochastic process models. We then introduce a more concrete graphical model called the latent Bernoulli process model. We show that even the problem of selecting the best object in the final step is #P-complete. To solve the crawling problem with a graphical structure, we propose a reinforcement learning-based algorithm, evaluate it, and compare it to strong baselines using real and synthetic data. Our experimental results show that the reinforcement learning-based algorithm outperforms the baselines. Additionally, we validate the intuition behind graph-structured crawling through experiments on real data.

ACM Reference Format:
Mohammadhossein Bateni, Lin Chen, Hossein Esfandiari, and Sasan Tavakkol. 2023. Graph-Structured Crawling: Model and Approach. In *Proceedings of Machine Learning on Graphs Workshop at WSDM'23 (MLoG at WSDM'23).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

*Authors are listed in alphabetical order.

1 INTRODUCTION

In traditional supervised active learning [6, 12, 18, 19], the learner aims to predict labels of examples, based on the previously queried and revealed labels. While the label may be stochastic, it is invariant in time, i.e., once it is drawn from the underlying distribution and added to the training or test dataset it does not change. In real world problems, there are many scenarios where the label can change over time. The crawling problem focuses on those with such renewal property [2–5, 14, 15, 17]. The main objective of crawling is to keep our knowledge up-to-date in an environment where information becomes outdated if not queried repeatedly. One important example of crawling is web crawling. In order to match up-to-date web content to a user's query, search engines need to maintain a cache of the latest versions of web pages even tough they constantly change. As an another example, consider an online geographical information platform (such as Google Maps, Yelp, etc.) that provides the opening hours of parks around the world and rely on crowdsourcing to obtain this information. These opening hours can change seasonally, due to holidays and unexpected events like the COVID-19 pandemic, and because of regular maintenance.

In both web crawling and detecting changes in opening hours, one needs to keep track of a large number of objects. In web crawling, these objects are the content of web pages. In detecting the changes in opening hours example, the objects are the opening hours of parks around the world. Previous studies often treat these objects as independent, but this is generally not true. For example, when breaking news occurs, the web pages of most mainstream media outlets report it, causing a group of news websites to update their pages at the same time. Similarly, during a pandemic, parks in a certain area are likely to change their hours together or at the beginning of winter, it is common for parks to change their hours. In Figure 1, we provide an example of how parks, which have strong connections with each other, may change their hours together during the winter season, while restaurants, which are not as strongly connected to parks but are strongly connected to each other, may not change their hours during the winter.

In this paper, we study the problem of crawling, i.e., keeping knowledge up to date where the information being studied may change over time. Crawling is relevant in a variety of contexts, including web crawling, where search engines seek to maintain a cache of the latest versions of web pages, and detecting changes in the opening hours of parks in a geographical information platform like Google Maps. In this work, we propose using a graph structure

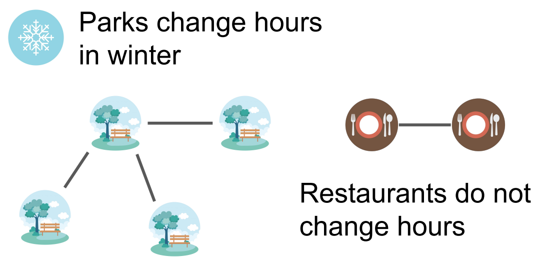


Figure 1: An example of how parks, which are strongly connected to each other, may change their hours collectively during the winter season, while restaurants, which are not as strongly connected to parks but are strongly connected among themselves, may not change their hours during the winter.

to model the interactions between the objects being studied in order to better capture the influence of one object on another in a crawling scenario. Our main contributions are as follows:

- We formally define the problem of graph-structured crawling, taking into account the correlation of updates of objects in a graph. To the best of our knowledge, this is the first work to consider the graphical structure underlying a crawling problem.
- We show that in the worst case it is impossible to learn anything interesting in polynomial time, while there exists a learnable structure hidden in the process that requires an exponential time to be discovered.
- We introduce the latent Bernoulli process model, which allows us to capture relationships between objects in a crawling scenario. However, we also demonstrate that even the problem of selecting the best object in the last step under this model is #P-complete. Additionally, We propose a dynamic programming solution for the crawling problem in a latent Bernoulli process model and provide insight on how to efficiently implement it through sampling.
- We verify the intuition behind graph-structured crawling that observing an update in an object increases the likelihood of finding updates in its neighboring objects.
- We propose a reinforcement learning-based (RL-based) algorithm to solve the crawling problem and show that it outperforms strong baselines on real and synthetic data.

The rest of the paper is organized as follows. In Section 2, we provide a review of related work on the crawling problem. In Section 3, we introduce a graphical model to capture the relationships between the objects being studied. In Section 4, we show an impossiblity result for the general problem. In light of this impossibility result, we introduce the latent Bernoulli process model in Section 5. We present the hardness of the problem of selecting the best object in the last step under this model in Section 5.1. To tackle the crawling problem with a graphical structure, we propose an algorithm based on reinforcement learning (RL) in Section 6 and evaluate it through experiments on real data in Section 7. Additionally, in Section 7 we also test the validity of the intuition behind graph-structured crawling using real data. Finally, Section 8 concludes the paper.

2 RELATED WORK

Cho and Garcia-Molina [7, 8] proposed various refresh policies and studied their effectiveness. The authors first formalized the notion of *freshness* of copied data by defining two freshness metrics and proposed a Poisson process as the change model of data sources. Based on this framework, the authors examined the effectiveness of the proposed refresh policies both analytically and experimentally. Cho and Garcia-Molina [9] focused on the problem of estimating the Poisson rate in the Poisson model that they employed to characterize webpage updates. Azar et al. [2] formally defined the optimization problem for finding a stationary policy for both discrete-time and continuous-time settings under the request rate model and the utility model. The goal was to determine the optimal request rate for each page to maximize the expected freshness. Han et al. [11] utilized five base crawling algorithms, including the one proposed by Azar et al. [2], and employed the multi-armed bandit method to determine which base algorithm to use at each step of the crawling process. Kolobov et al. [15] proposed an extension to the results of Azar et al. [2], which considered the politeness constraint on the rate at which pages can be requested from a given host. For a survey of web crawling, see the work of Olston et al. [17].

While the title of [22] may imply a similar problem, the authors actually studied a different problem. They focused on how to crawl and save an online social network in order to collect a large amount of information such as contact lists, profiles, pictures, and videos. The main difference between their work and the crawling problem that we are interested in is that they aimed to create a mirror of an online social network and save it to, say, a hard drive, while our focus is on tracking updates to online content. In [10], the focus was on a different problem involving an agent (referred to as a spider) moving between nodes in a randomly growing web graph.

3 GRAPH-STRUCTURED CRAWLING

In this section, we introduce the concept of graph-structured crawling. Specifically, we formalize the correlations between the updates of objects in a graph and define the crawling problem on such graphs.

Updates of graph-structured objects. We use a counting process to model the updates over time. Recall that a counting process $\{N(t)\}_{t\geq 0}$ is a stochastic process such that $N(0) = 0$ and $N(t)$ is a right-continuous step function with jumps of size +1. Each jump of $N(t)$ represents an update.

Let V denote the set of objects of interest, and $\{N_v(t)\}$ be the counting process of an object $v \in V$. We assume that the joint stochastic process $\{N_V(t)\}_{t\geq 0} \triangleq \{N_v(t)\}_{t\geq 0, v\in V}$ has an (unknown) joint distribution $\Pr(N_v(t) : v \in V)$, where a graphical model can be introduced.

For example, we may assume that $\Pr(N_V(t))$ is a Markov random field with its joint probability density being strictly positive [13]. In other words, if the objects form a graphical structure $G = (V, E)$, then (by the clique factorization characterization by the Hammersley-Clifford theorem)

$$\Pr(N_V(t)) \propto \prod_{C \in \mathrm{Cl}(G)} \phi_C(N_C(t)), \tag{1}$$

where $\mathrm{Cl}(G)$ is the set of cliques of G, $N_C(t) = \{N_v(t)\}_{v \in C}$, and ϕ_C is called the clique potential.

For another example, we may also assume that $\Pr(N_V(t))$ is a Bayesian network.

Crawling. In the crawling problem, a crawler interacts with an environment consisting of a joint stochastic process $\{N_V(t)\}_{t \geq 0}$. At each query time t_i ($i = 1, 2, 3, \ldots$ and $0 \leq t_1 < t_2 < \cdots$)[1], the crawler selects an object $v_i \in V$ and observes whether it has changed since the last time the crawler queried it. The crawler's observation at time t_i is given by

$$o_i = \mathbb{1}\left\{N_{v_i}(A(v_i, t_i)) \neq N_{v_i}(t_i)\right\}. \tag{2}$$

where $A(v, t)$ is the time of the last query on v prior to time t (or 0 if v_i has never been queried before), and is formally defined by

$$A(v, t) \triangleq \max\left(\{0\} \cup \{t_j \mid v_j = v, t_j < t\}\right). \tag{3}$$

The crawler's reward at time t_i is determined by the reward function $r_i = r(v_i, t_i)$, which could either be equal to o_i if the crawler only cares about whether it hits an update at time t_i, or

$$r_i = f\left(\left(t_i - \lceil A(v_i, t_i) \rceil_{N_{v_i}(\cdot)}\right)_+\right) \tag{4}$$

if the crawler also takes into account the freshness of object v_i at time t_i, where $(x)_+ = \max\{x, 0\}$, f is a non-negative non-increasing function, $\lceil \tau \rceil_{N_v(\cdot)} \triangleq \sup\{t \in (\tau, \infty) \mid N_v(t) = N_v(\tau)\}$ is the next jumping time of $N_v(\cdot)$ at time τ.

The aim of the crawler is to maximize either the expected average return over T rounds, $\mathbb{E}[\mathsf{AR}_T]$, if the game lasts for a fixed number of rounds, or the expected limiting average return, $\mathbb{E}[\mathsf{AR}_\infty]$, if the game is ongoing indefinitely. The T-round average return is defined as $\mathsf{AR}_T = \frac{1}{T}\sum_{i=1}^{T} r_i$, and the limiting average return is defined as $\mathsf{AR}_\infty = \liminf_{T \to \infty} \mathsf{AR}_T$.

4 AN IMPOSSIBILITY RESULT

In this section, we present a novel fundamental impossibility result in crawling. Specifically, we demonstrate that in the worst case, it is not possible to learn anything meaningful in polynomial time, despite the presence of a learnable structure that requires exponential time to uncover. Mathematically, we prove that there exists a process model such that $\mathbb{E}[\mathsf{AR}_T] \in o(1)$ for any polynomial T, while $\mathbb{E}[\mathsf{AR}_\infty] = 1$. This result motivates us to investigate specific stochastic process models in the subsequent sections.

Theorem 4.1. *Let us set the reward function*

$$r_i = f\left(\left(t_i - \lceil A(v_i, t_i) \rceil_{N_{v_i}(\cdot)}\right)_+\right) \tag{5}$$

as in equation 4 , where $f(t) = e^{-t}$. For any algorithm, there exists a process (unknown to the algorithm) over n objects such that $\mathbb{E}[\mathsf{AR}_T] = o(1)$ for any polynomial T, while $\mathbb{E}[\mathsf{AR}_\infty] = 1$.

[1]The crawler can choose the query time t_i adaptively, potentially subject to a constraint such as a minimum time between queries ($t_{t+1} - t_i \geq 1$), or the query times can be pre-determined.

Proof. Pick a number k such that $k \in o(\sqrt{n})$ and $k \in \omega(\log n)$. Without loss of generality we assume that n is divisible by k. First we define a mapping $\psi : 2^{n/k} \to 2^{n/k}$ then we use this mapping to define our process. To define ψ, for each binary number x with length n/k, we generate a random binary number y of length n/k where k of the bits of y chosen uniformly at random are 1 and the rest of the bits are 0, and we set $\psi(x) = y$. We also define a function $\pi : 2^{n/k} \to k^k$, which maps each binary number x of length n/k to a random permutation of length k. We use $\pi_i(x)$ to refer to the i-th element in the random permutation $\pi(x)$. Note that even though ψ and π are defined through a random process, after defining them, each $x \in 2^{n/k}$ is mapped to a fixed $\psi(x)$ and a fixed $\pi(x)$.

Now we are ready to define our process. Our process consists of some phases and each phase consists of k time-steps. We first define how our phases evolve and then explain how the objects are updated in each phase. Each phase corresponds to a binary number of length n/k. The first phase corresponds to all zeros. If a phase corresponds to x the next phase corresponds to $\psi(x)$. Note that the probability of observing a number twice, (i.e., fall in a loop) in the first $\binom{n/k}{k}^{0.4}$ phases is at most $\frac{\binom{n/k}{k}^{0.4} \times \binom{n/k}{k}^{0.4}}{\binom{n/k}{k}} = \binom{n/k}{k}^{-0.2} \in o(1)$. In the rest we upper-bound $\mathbb{E}[\mathsf{AR}_T]$ while we are not in a loop.

Now we explain how the objects are updated in each phase. We refer to the objects as $u_{i,j}$ where $1 \leq i \leq k$, and $1 \leq j \leq n/k$. In the i-th time-step of phase x, we set $j_{x,i}$ to the index of the $\pi_i(x)$-th bit with value 1 in x and update the object $u_{i,j_{x,i}}$. Note that $j_{x,i}$ is equivalent to a number chosen uniformly at random form $\{1, \ldots, n/k\} \setminus \{j_{x,1}, \ldots, j_{x,i-1}\}$. Hence, conditioned on the past, the probability that the algorithm queries the updated object $u_{i,j_{x,i}}$ is at most $\frac{1}{n/k-i}$. Similarly, the probability that the algorithm queries an object that is updated in the past k time-steps is at most $\frac{1}{n/k-i-k} \leq \frac{1}{n/k-2k} \in O\left(\frac{1}{\sqrt{n}}\right) \in o(1)$. Moreover, if the algorithm queries an object that is updated more than k time-steps before, its reward is $e^{-k} \in o(1)$. Therefore, while the phases do not fall in a loop, the expected reward of each query in $o(1)$ and hence $\mathbb{E}[\mathsf{AR}_T] = o(1)$ as claimed.

Moreover, we know that there are $\binom{n/k}{k}$ different types of phases. Hence if we query an object $2k\binom{n/k}{k}$ times, we observe that it falls in a loop. Hence after $2nk\binom{n/k}{k}$ queries, we exactly know when each object is updated. Since in each time-step exactly one object is updated, after $2nk\binom{n/k}{k}$ queries, we can collect 1 reward per query and hence we have $\mathbb{E}[\mathsf{AR}_\infty] = 1$, which completes the proof. □

5 LATENT BERNOULLI PROCESS MODEL

In this section, we introduce the latent Bernoulli process model to capture various properties observed in real-world crawling scenarios.

Discrete Time. To facilitate practical computation, we assume that time is discrete, with updates to $N_v(t)$ occurring only at integer-valued time steps.

Latent Bernoulli Process. A latent Bernoulli process $M_v(t)$ is associated with each object v and is only updated at integer-valued time steps. At each time step $t \in \mathbb{N}$, an update to $M_v(t)$ occurs with probability $\lambda_v(t) \in [0, 1]$. The latent processes $M_v(t)$ are not

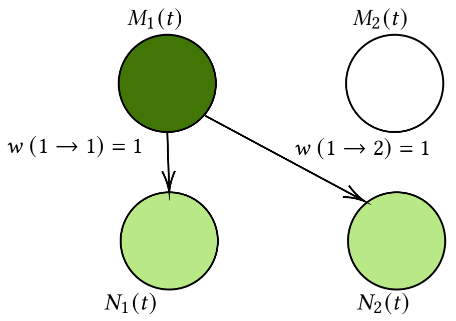


Figure 2: Example of a latent Bernoulli process.

observable to us. The Bernoulli probability $\lambda_v(t)$ can be interpreted as our prior belief that the object v will experience an update at time t.

Related objects update simultaneously. The graph structure is directed and weighted, and has self-loops. The weight $w(u \longrightarrow v) \in [0,1]$ is the probability of the latent Bernoulli process $M_u(t)$ influencing (transferring an arrival to) $N_v(t)$. In other words, once there is an arrival of $M_u(t)$, the process $N_v(t)$ receives an arrival with probability $w(u \longrightarrow v)$. Formally, we have

$$\Delta N_v(t) = \bigvee_u B_{u\rightarrow v}(t)\Delta M_u(t)\,, \tag{6}$$

where $\Delta N_v(t) = N_v(t) - N_v(t-1)$, $\Delta M_u(t) = M_u(t) - M_u(t-1) \sim \mathrm{Ber}(\lambda_u(t))$, and $B_{u\rightarrow v}(t) \sim \mathrm{Ber}(w(u \longrightarrow v))$.

Note that this is *not* a diffusion process over a graph. See an example in Figure 2, where $N_1(t)$ and $N_2(t)$ have arrival whenever $M_1(t)$ has an arrival. The processes $N_1(t)$ and $N_2(t)$ receive this arrival *simultaneously*. There is no diffusion from node 1 to node 2. This captures the situation where related objects have an update simultaneously.

Seasonality. So far, we have not specified the Bernoulli probabilities of $\lambda_v(t)$ of the latent processes $M_v(t)$. The simplest assumption is that $\lambda_v(t) = \lambda_v$ is constant for each object v. However, this cannot capture the seasonality of the environment of real-world crawling problems. As we motivated in the introduction, parks usually change their hours at the beginning of every summer and winter. To this end, we assume that $a_v(t) = \log \frac{\lambda_v(t)}{1-\lambda_v(t)} \in \mathbb{R}$ (therefore $\lambda_v(t) = \frac{1}{1+\exp(-a_v(t))}$ is the logistic function of $a_v(t)$) follows an autoregressive process AR(p) [20]:

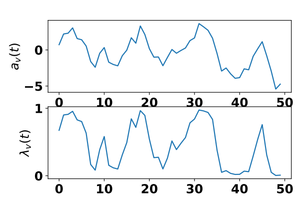
$$a_v(t) = \phi_v(0) + \sum_{i\in[p]} \phi_v(i) a_v(t-i) + \epsilon_v(t)\,, \tag{7}$$

where $\{\epsilon_v(t)\}$ is white noise. Figure 3 illustrates an example of AR(7) process whose period is 7.

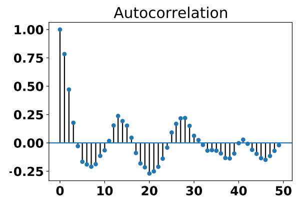
Observation and reward. We define the reward function

$$r(v,t) = \mathbb{1}\{N_v(A(v,t)) \neq N_v(t)\} \tag{8}$$
$$= \bigvee_{i\in(A(v,t),t]\cap\mathbb{Z}} \bigvee_u B_{u\rightarrow v}(i)\Delta M_u(i)\,. \tag{9}$$

The observation of action v_t at time t is $r(v_t, t)$.



(a) $a_v(t)$ and $\lambda_v(t)$



(b) Autocorrelation function

Figure 3: An AR(7) process.

5.1 Hardness Result

In this subsection, we demonstrate the hardness of the problem of selecting the optimal object, even in the final step, under the latent Bernoulli process model. Specifically, we prove that it is #P-complete by reducing it to the counting problem of vertex cover.

The SelectBestObject (SBO) Problem is given the input:

- A set of weights $w(u \longrightarrow v)$ for each pair of objects $u, v \in V$,
- A set of parameters $\lambda_v(t)$ for each object v and time $t \in [T+1]$,
- Queries v_t made at each time $t \in [T]$,
- Observations $r(v_t, t)$ made at each time $t \in [T]$

SBO involves finding the object that maximizes the expected value of its reward at time $T+1$

$$\arg\max_{v\in V} \mathbb{E}\left[r(v, T+1) \mid r(v_t,t), t\in[T]\right]\,. \tag{10}$$

Even when the weights $w(u \longrightarrow v)$ are binary and the observations $r(v_t,t)$ are fixed at 1, the SBO problem is hard to solve.

Theorem 5.1. *The SBO problem is #P-complete, even when $w(u \longrightarrow v) \in \{0,1\}$ for every pair of objects u, v and $r(v_t,t) = 1$ for all $t \in [T]$.*

Proof. **Step 1:** Show that SBO is not easier than computing the value of

$$\max_{v\in V} \mathbb{E}\left[r(v, T+1) \mid r(v_t,t), t\in[T]\right]\,. \tag{11}$$

We call the problem of computing the value of equation 11 the ComputeBestReward (CBR) problem. We shall show this by reducing CBR to SBO. We augment the problem by adding a new object $v^\dagger$ to V, resulting in an augmented object set $V^\dagger = V \cup \{v^\dagger\}$. We set $w(v^\dagger \longrightarrow v^\dagger) = 1$, $\lambda_{v^\dagger}(1) = \lambda_{v^\dagger}$, and $\lambda_{v^\dagger}(t) = 0$ for $\forall t = 2,3,\ldots,T+1$. Then we have $\mathbb{E}\left[r(v^\dagger, T+1) \mid r(v_t,t), t\in[T]\right] = \lambda_{v^\dagger}$. We have $\arg\max_{v\in V^\dagger} \mathbb{E}\left[r(v,T+1) \mid r(v_t,t), t\in[T]\right] = v^\dagger$ if and only if $\lambda_{v^\dagger} > \max_{v\in V} \mathbb{E}\left[r(v,T+1) \mid r(v_t,t), t\in[T]\right]$. By varying $\lambda_{v^\dagger}$ (e.g., binary search), we can compute equation 11 up to an arbitrary precision.

Step 2. Reduction from ComputeBernoulliPosterior (CBP) to CBR. We define the CBP problem below.

- Input: $A \in \{0,1\}^{m\times n}$ and $b \in \{0,1\}^n$.
- Output: compute $\Pr_{x\sim \mathrm{Ber}(1/2)^{\otimes n}}[b^\top x \geq 1 \mid Ax \geq \mathbf{1}]$ where $\mathrm{Ber}(1/2)^{\otimes n}$ is the distribution of a random vector whose entries are independent $\mathrm{Ber}(1/2)$ random variables.

We create $\max\{m,n\}+1$ objects $V = \{u_0, u_1, \ldots, u_{\max\{m,n\}}\}$ and set $T = m$.

We set $w(u_j \to u_i) = A_{i,j}$ for all $i \in [m], j \in [n]$, and set $w(u_j \to u_0) = 1$ if and only if $b_j = 1$. The w values of all other directed edges are all zero.

Moreover, set $\lambda_{u_i}(1) = 1/2$ for $\forall i \in [n]$, set $\lambda_u(1) = 0$ for $\forall u \in V \setminus \{u_i\}_{i\in[n]}$, and set $\lambda_u(t) = 0$ for $\forall t = 2, 3, \ldots, T+1$ and $\forall u \in V$. We set $v_t = u_t$ and $r(v_t, t) = 1$ for $t \in [m]$.

At time $T+1$, with probability 1, u_i ($i \in [m]$) has no update since last query at time i (this is because $\lambda_u(i) = 0$ for $\forall i = 2, 3, \ldots, T+1$ and $\forall u \in V$). Any other object u_i ($i > m$) has never been queried and has no update since the very beginning with probability 1 (this is because $w(u \to u_i) = 0$ for all $i > m$ and $u \in V$. Therefore, all objects $\{u_i\}_{i\geq 1}$ will have reward 0 at time $T+1$, and thus none of them is optimal. Object u_0 has never been queried either, and we have

$$\mathbb{E}\left[r(u_0, T+1) \mid r(v_t, t), t \in [T]\right] = \Pr[b^\top x \geq 1 \mid Ax \geq \mathbf{1}] . \quad (12)$$

Step 3. We reduce the problem of computing $\Pr_{x\sim \mathrm{Ber}(1/2)^{\otimes n}}[Ax \geq \mathbf{1}]$ to CBP, where $\mathrm{Ber}(1/2)^{\otimes n}$ represents the probability distribution of n-dimensional binary random vectors whose entries are i.i.d. $\mathrm{Ber}(1/2)$. It suffices to decompose $\Pr_{x\in \mathrm{Ber}(1/2)^{\otimes n}}[Ax \geq \mathbf{1}]$ into the product of conditional probabilities

$$\Pr[Ax \geq \mathbf{1}] = \Pr[a_1^\top x \geq \mathbf{1}] \Pr[a_2^\top x \geq \mathbf{1} \mid a_1^\top x \geq \mathbf{1}] \quad (13)$$

$$\cdots \Pr[a_m^\top x \geq \mathbf{1} \mid A_{1:m-1,:} x \geq \mathbf{1}] , \quad (14)$$

where $a_i^\top$ is the i-th row of A and $A_{1:m-1,:}$ is the submatrix of first $m-1$ rows of A.

Step 4. We reduce the counting problem of vertex covers in an undirected graph, $\tilde{G} = (\tilde{V}, \tilde{E})$, to the problem of computing $\Pr_{x\in \mathrm{Ber}(1/2)^{\otimes n}}[Ax \geq \mathbf{1}]$. To do this, we set $m = |\tilde{E}|$ and $n = |\tilde{V}|$ (thus $A \in \mathbb{R}^{|\tilde{E}|\times|\tilde{V}|}$), and for every edge $e \in \tilde{E}$ and vertex $v \in \tilde{V}$, we define $A_{e,v} = 1$ if and only if vertex v is incident to edge e, and set it to 0 otherwise. □

5.2 Dynamic Programming

In this subsection, we present a dynamic programming algorithm in Algorithm 1 for solving the web crawling problem. The input for this algorithm is a set of previously queried objects and their corresponding rewards up to time $t-1$, and its objective is to determine the optimal object to query for the t-th step. By reducing the problem for the t-th step to the problem for the $(t+1)$-th step, it breaks the problem down into smaller subproblems. The algorithm can be applied by working backwards from a solution at the final time step T to solve the problem for each step up to t, as long as the horizon T is finite.

Algorithm 1 DynamicProgramming (DP)

```
Input: $\{(a_i, r_i) | 1 \leq i \leq t-1\}$
1: $H_{t-1} \leftarrow \{(a_i, r_i) | 1 \leq i \leq t-1\}$
2: $p_j(v) \leftarrow \Pr(r_t = j \mid H_{t-1}, v_t = v), \forall j \in \{0,1\}$
3: if $t < T$ then
4:     $q_j(v) \leftarrow \mathrm{DP}(H_{t-1}, v_t = v, r_t = j), \forall j \in \{0,1\}$
5: else
6:     $q_j(v) \leftarrow 0, \forall j \in \{0,1\}$
7: end if
8: $v \leftarrow \arg\max_{v\in V} p_1(v)(1 + q_1(v)) + p_0(v)(0 + q_0(v))$
9: return $v$ and the maximum value
```

The randomness of the graph-structured process $\{N_v(t)\}_{v\in V}$ is determined by the Bernoulli random variables $\Delta M_u(t) = M_u(t) - M_u(t-1) \sim \mathrm{Ber}(\lambda_u(t))$ and $Bu \to v(t) \sim \mathrm{Ber}(w(u \to v))$ for all nodes u, v in the graph and for all times $t \in [T]$. There are a total of $O(|V|^2 T)$ such Bernoulli random variables. As a result, the computation of $p_j(v)$ in Algorithm 1 requires summing over all possible combinations of these Bernoulli random variables, and therefore requires an exponential time complexity. Then the DP algorithm is known to take exponential time as computing the conditional probability in Algorithm 1 is an exponential process. However, one can estimate the conditional probability by sampling. The estimation will be accurate due to the concentration inequality such as McDiarmid's inequality because changing any one of the Bernoulli random variables can result in only a small change in the conditional probability. By taking the estimation error into account, we can develop an approximation algorithm based on both the dynamic programming and sampling techniques.

6 REINFORCEMENT LEARNING-BASED CRAWLER

In this section, we propose a reinforcement learning-based (RL-based) algorithm to solve the crawling problem. The algorithm takes advantage of the central intuition that, once we observe an update in an object, it is likely that we will also observe updates in its neighbors. Therefore, we encode the previous observations of the crawler into a state, and based on this state, the RL-based crawler seeks to find the best object to query.

Designing history-encoding state. To design our state space, we represent each object in the set V as a row in a two-dimensional array of shape $|V| \times 5$. Thus the state space is $\mathcal{S} = (\{0,1\} \times \mathbb{N} \times \mathbb{N} \times \mathbb{R} \times \mathbb{R})^V$. The first column is binary and indicates whether an object has been queried since the beginning of the crawling process (0 if not queried, 1 if queried). The first column is initialized to all zeros. The second column is a positive integer that denotes the number of time units since the object was last queried and the outcome was *updated.* Similarly, the third column is a positive integer that denotes the number of time units since the object was last queried and the outcome was *not updated.* The fourth and fifth columns store historical observations, possibly with a decay factor (whose value is between 0 and 1) applied over time. At each step, if an object is queried and *updated,* its corresponding fourth column is set to 100. If the object is queried but *not updated* its corresponding fifth column is set to 100. If the object is not queried at all, the values stored in its corresponding fourth and fifth columns are multiplied by the decay factor.

At each step of the crawling process, the crawler selects an object from the set V to query for updates. The set of possible actions $\mathcal{A}$, represented as the action space, is equal to V.

After choosing an object to query, the crawler observes the outcome, which is binary (1 if the object has been updated since the last query, 0 otherwise). As previously mentioned, the transition of the state occurs as follows: the first column of the queried object is set to 1, and if the outcome is *updated*, the second column is set to 1 and the fourth column is set to 100, while the third column is incremented by 1. If the outcome is *not updated*, the second column is incremented by 1 and the fifth column is set to 100, while the third column is set to 1. For all other objects that were not queried at this step, their second and third columns are incremented by 1. It is important to note that the fourth and fifth columns of all objects experience natural decay at the beginning of every step.

The policy of the crawler, denoted as $\pi(v \mid s)$, represents the probability of choosing object v at state s. Given the initial state s_1 and the policy π, we can define the probability distribution $D^{\pi}(\tau)$ of the trajectory $\tau = (s_1, v_1, r_1, s_2, v_2, r_2, \dots)$, where s_i is the state at the ith step, v_i is the object chosen by the crawler at the ith step, and r_i is the reward received at the ith step by choosing v_i. The return $R(\tau)$ for a given trajectory τ is defined as the sum of all rewards received along the trajectory, with each reward being discounted by a factor of γ raised to the power of the number of steps since it was received: $R(\tau) = \sum_{i \geq 0} \gamma^i r_i$, where $\gamma \in (0, 1)$. The objective of the RL-based algorithm is to maximize the expected return $\mathbb{E}_{\tau \sim D^{\pi}} [R(\tau)]$.

In our proposed RL-based approach, we utilize the double Deep Q-Network (double DQN) algorithm [21]. The discount value is set to $\gamma = 0.7$, and we use a two-layer neural network with widths of 75 and 40 as our Q network and the target Q network.

7 NUMERICAL STUDIES

7.1 Datasets

We conduct experiments on the Beijing Multi-Site Air-Quality Dataset [23] to evaluate the performance of the proposed RL-based algorithm and verify the intuition behind graph-structured crawling. The dataset includes hourly air pollutant readings from 12 monitoring sites in Beijing, as well as matched meteorological data from the China Meteorological Administration. The data covers the period from March 2013 to February 2017, and we are interested in tracking changes in the PM10, CO, and PM2.5 columns. In our crawling scenario, there is an update every time the values for PM10, CO, or PM2.5 exceed certain thresholds (800, 8000, and 100 respectively). In this dataset, each object represents an air quality monitoring site in Beijing. The goal of the crawling algorithm is to visit each site at each step and capture any changes in the values of PM10, CO, and PM2.5 at that site. The algorithm must be able to detect and record any updates to these values to maintain an accurate and up-to-date understanding of the air quality in Beijing.

We also examine the proposed RL-based method on the latent Bernoulli model using a graph structure generated by an Erdős-Rényi model with $|V| = 10$ nodes and an edge probability of $3/(|V| - 1) = 1/3$. The probability of transferring updates from the latent Bernoulli process to the exposed process is fixed and sampled from a Beta distribution $\text{Beta}(7, 3)$. Additionally, for neighboring nodes, the probability of transferring updates is also fixed and sampled from a Beta distribution $\text{Beta}(3, 15)$. The rate of the latent Bernoulli process is fixed and sampled from a Beta distribution $\text{Beta}(2, 30)$.

7.2 Verification of Intuition behind Graph-Structured Crawling

We aim to verify the intuition behind graph-structured crawling, which suggests that updates in an object's neighbors are likely to occur following an update in the object. To test this, we use the Beijing Multi-Site Air-Quality Dataset. For each pair of monitoring sites u and v, we measure the delay in transferring an update from

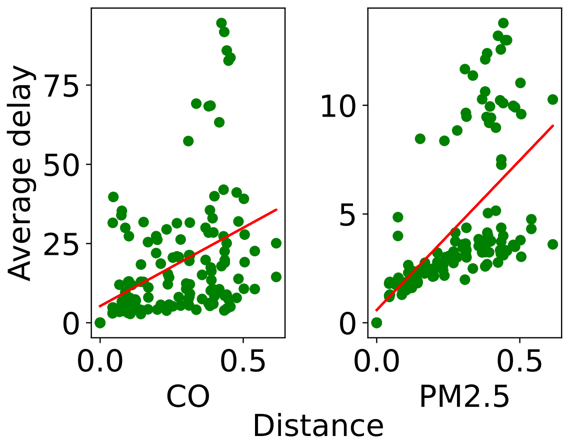


Figure 4: A scatter plot illustrating the correlation between the average delay in transferring updates and the distance between monitoring sites in the Beijing Multi-Site Air-Quality Dataset. The left and right panels depict the results for updates of CO and PM2.5 respectively, when the threshold is exceeded. A regression line is also shown, which demonstrates a positive correlation between the distance and average delay, thus supporting the intuition behind graph-structured crawling.

u to v by calculating the time difference between an update at u and the next update at v. We then compute the average delay for each pair of sites and plot a scatter plot in Figure 4, with the horizontal coordinate representing the distance between the sites and the vertical coordinate representing the average delay. A regression line is also plotted. The results show a positive correlation between distance and average delay, indicating that updates in closer sites are more likely to occur simultaneously.

7.3 Performance Evaluation of Crawling Algorithms

We compare the RL-based algorithm to two baseline algorithms: Exp3 [1, 16], an adversarial bandit algorithm, and a variant of the active covering algorithm (AC) [12]. The original active covering algorithm is not suitable for use in the crawling problem, as it assumes that the label does not change once it has been revealed. However, in a crawling scenario, the label may change whenever there is an update. Therefore, we modified the original active covering algorithm to better suit the needs of the crawling problem by taking into account the possibility of label changes. We implemented a periodic restart of the active covering algorithm once all objects have been queried. At the time of the restart, all objects are marked as unqueried and the algorithm begins from the start. This allows us to continually update our knowledge of the objects as new information becomes available.

To evaluate the performance of the RL-based algorithm and the baselines, we calculated their hitting rates. The hitting rate for a crawling algorithm is the percentage of queries that result in an update to a site since the algorithm's last query of that site. This

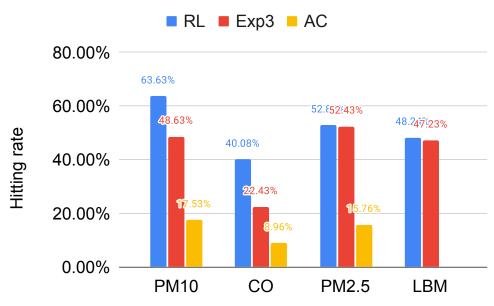


Figure 5: The chart illustrates the performance comparison of the RL-based algorithm with the baselines in terms of hitting rate. The LBM column represents the results for the latent Bernoulli model.

Task	Algorithm	Hitting rate	Std
PM10	RL	**63.634%**	**0.353%**
	Exp3	48.63%	1.342%
	AC	17.528%	0.419%
CO	RL	**40.079%**	0.523%
	Exp3	22.426%	1.366%
	AC	8.96 %	**0.434%**
PM2.5	RL	**52.847%**	**0.198%**
	Exp3	52.426%	0.533%
	AC	15.762%	0.627%
LBM	RL	**48.238%**	**0.142%**
	Exp3	47.226%	0.889%

Table 1: Hitting rate and standard deviation of the RL-based algorithm and the baselines. The LBM rows represent the results for the latent Bernoulli model.

metric allows us to measure the efficiency of the algorithms in detecting and recording changes in the objects being studied. A higher hitting rate indicates that the algorithm is more effective at discovering updates and maintaining an up-to-date understanding of the objects. We present the results of our experiments in Figure 5 and provide more detailed numerical results, including standard deviations, in Table 1.

In all three tasks of tracking PM10, CO, and PM2.5, the RL-based algorithm consistently outperforms the other algorithms in terms of hitting rate. In the tasks of tracking PM10 and CO, the RL-based algorithm shows a particularly strong advantage, with hitting rates of 63.6% and 40.1% respectively. This is significantly higher than the hitting rates achieved by Exp3, which are 48.6% for PM10 and 22.4% for CO. For PM10, the RL-based algorithm is 31% better than Exp3, and for CO, it is 79% better. In addition, the standard deviation of the RL-based algorithm is smaller than that of Exp3 in both tasks. The smaller standard deviations of the RL-based algorithm also suggest that it is more stable and reliable than the other algorithms. In the task of tracking PM2.5, while the hitting rates of the RL-based algorithm and Exp3 are closer, the RL-based algorithm still outperforms Exp3 by roughly one standard deviation. Its standard deviation is again smaller than that of Exp3. Both the RL-based algorithm and Exp3 significantly outperform the variant of the active covering algorithm in this task. The results for the latent Bernoulli model also show that the RL-based algorithm outperforms Exp3 in terms of both the hitting rate and the standard deviation of the hitting rate.

8 CONCLUSION

In this study, we addressed the problem of crawling in dynamic environments, where labels or information can change over time. We proposed using a graph structure to model the relationships between objects to better capture influence in a crawling scenario. We showed that it is impossible to learn in polynomial time in worst case, and a exponential time algorithm is needed to find the structure. We then proposed a reinforcement learning-based algorithm and evaluated its performance on real and synthetic data, showing its effectiveness in solving the crawling problem. We also verified the intuition behind graph-structured crawling using real data.

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 1995. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *Proceedings of IEEE 36th annual foundations of computer science*. IEEE, 322–331.

[2] 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.

[3] Melih Bastopcu and Sennur Ulukus. 2020. Information freshness in cache updating systems. *IEEE Transactions on Wireless Communications* 20, 3 (2020), 1861–1874.

[4] Melih Bastopcu and Sennur Ulukus. 2020. Who should Google scholar update more often?. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 696–701.

[5] Melih Bastopcu and Sennur Ulukus. 2021. Age of information for updates with distortion: Constant and age-dependent distortion constraints. *IEEE/ACM Transactions on Networking* 29, 6 (2021), 2425–2438.

[6] Lin Chen, Hamed Hassani, and Amin Karbasi. 2017. Near-optimal active learning of halfspaces via query synthesis in the noisy setting. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[7] Junghoo Cho and Hector Garcia-Molina. 2000. Synchronizing a database to improve freshness. *ACM sigmod record* 29, 2 (2000), 117–128.

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

[9] Junghoo Cho and Hector Garcia-Molina. 2003. Estimating frequency of change. *ACM Transactions on Internet Technology (TOIT)* 3, 3 (2003), 256–290.

[10] Colin Cooper and Alan Frieze. 2002. Crawling on web graphs. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 419–427.

[11] 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*. 407–417.

[12] Heinrich Jiang and Afshin Rostamizadeh. 2021. Active Covering. In *International Conference on Machine Learning*. PMLR, 5013–5022.

[13] Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

[14] Andrey Kolobov, Sébastien Bubeck, and Julian Zimmert. 2020. Online learning for active cache synchronization. In *International Conference on Machine Learning*. PMLR, 5371–5380.

[15] 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.

[16] Tor Lattimore and Csaba Szepesvári. 2020. *Bandit algorithms*. Cambridge University Press.

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

[18] Burr Settles. 2012. Active learning. *Synthesis lectures on artificial intelligence and machine learning* 6, 1 (2012), 1–114.

[19] Simon Tong. 2001. *Active learning: theory and applications.* Stanford University.

[20] Ruey S Tsay. 2014. *An introduction to analysis of financial data with R.* John Wiley & Sons.

[21] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 30.

[22] Shaozhi Ye, Juan Lang, and Felix Wu. 2010. Crawling online social graphs. In *2010 12th International Asia-Pacific Web Conference*. IEEE, 236–242.

[23] Shuyi Zhang, Bin Guo, Anlan Dong, Jing He, Ziping Xu, and Song Xi Chen. 2017. Cautionary tales on air-quality improvement in Beijing. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 473, 2205 (2017), 20170457.