Efficient Crawling for Scalable Web Data Acquisition

Antoine Gauquier
antoine.gauquier@ens.psl.eu
DI ENS, ENS, CNRS, PSL Univ., Inria
Paris, France

Ioana Manolescu
ioana.manolescu@inria.fr
Inria & Institut Polytechnique de Paris
Palaiseau, France

Pierre Senellart
pierre@senellart.com
DI ENS, ENS, CNRS, PSL Univ., Inria
Paris, France

Abstract

Journalistic fact-checking, as well as social or economic research, require analyzing *high-quality statistics datasets* (*SDs*, in short). However, retrieving SD corpora at scale may be hard, inefficient, or impossible, depending on how they are published online. To improve open statistics data accessibility, we present a *focused Web crawling algorithm* that retrieves as many *targets*, i.e., resources of certain types, as possible, from a given website, in an efficient and scalable way, by crawling (much) less than the full website. We show that optimally solving this problem is intractable, and propose an approach based on reinforcement learning, namely using sleeping bandits. Our crawler efficiently learns *which hyperlinks lead to pages that link to many targets*, based on the paths leading to the links in their enclosing webpages. Our experiments on websites with millions of webpages show that our crawler is *highly efficient*, delivering high fractions of a site's targets while crawling only a small part.

1 Introduction

Open Data designates datasets whose use is open to all, without access fees or legal restrictions. Open Data has long been shared through the Web; in particular, many HTML pages contain *entity-centric tables*, where each line is about an entity, e.g., album, and describes its attributes, such as artist or year. Entity-centric tables have been leveraged to extract large-scale, open knowledge bases, e.g., [4, 37, 50, 53], and for question answering, e.g., [13, 29, 30, 52]. They are also frequent in *data lakes*, e.g., [12, 18, 21].

Statistics datasets (**SDs**, in short) are a distinct, highly useful type of Open Data. They are compiled by domain specialists typically working for national or international governing bodies, such as the United Nations, the International Monetary Fund, but also by companies, NGOs, etc. Some organizations, e.g., Eurostat, publish hundreds of thousands of SDs, updated yearly; others, e.g., an NGO focused on equitable justice, or a pollution watchdog in a certain area, may only publish a few dozen highly specialized datasets. **SDs significantly differ from entity-oriented Web tables**. SD content is mostly **numeric**, e.g., birth and death counts by age in a country over decades, or chip production per country and type of chip. From a data management perspective, SDs are **multidimensional aggregates**, sometimes **data cubes**. Also, SDs are mostly published as standalone files, encoded in a variety of formats, e.g., CSV, TSV, spreadsheets, XML dialects such as SDMX [46], JSON, etc. Open SDs are also sometimes embedded in PDF documents.

Answering queries or questions over tables, and in our context, over SDs, is as the very core of the data management science and industry. The wide availability of statistic datasets on the Web requires tools that make such data reachable in order to realize its potential; as we show in the paper, such tools are currently lacking.

SDs are of **great public utility**. Officials rely on them to inform decisions and support debates; social scientists use them to analyze policy impacts and detect trends; newsrooms use them to check the accuracy of public statements, make comparisons across countries, etc. Also, the *statistical literacy* of the general public needs to be improved: for instance, US voters vastly misrepresent the size of various fractions of the US population, e.g., how many are transgender (estimated: 21%, true: 0.6%) [42]. Thus, *building and using warehouses of statistics data* is central. Research works seeking to automate the recognition and checking of claims leveraging on retrieved corpora of SDs include, e.g., [2, 6, 9, 32, 44] in the Information Retrieval and Data Management, and [51] in NLP.

Finding all SDs published by an organization is desirable in order to study them as a whole, compare among organizations, populate a data lake or a database for fact-checking (as above), etc. For example, a work historian may want to retrieve **all SDs published by the International Labor Organization** (ILO); a journalist may need **all Eurostat SDs on poverty**. Unfortunately, current methods for this are either lacking, or quite inefficient.

Some websites publish an API giving access to all their SDs, e.g., Eurostat, Chicago Data Portal, yet writing custom code for each such API is cumbersome and brittle when APIs change. Worse, many sites hosting numerous Open SDs, e.g., the ILO, do not provide APIs to access all their data. *Search engines* (**SEs**, in short) and associated data portals turn out to provide access to **only a tiny fraction** of existing SDs (see Sec. 4.2); also, how these are selected is opaque to users. A **naïve, exhaustive website crawl is extremely inefficient** for large websites: (i) acquiring a full website takes space and time; (ii) *crawling ethics* requires to wait (typically around 1 second) between two successive HTTP requests; for a site of 1 million pages, such waits, alone, take more than 11 days. Thus, we are interested in a **focused crawl**, seeking to acquire within a given website as many **targets** as possible, while **minimizing the resources consumed**, e.g., HTTP requests or volume of transferred data. Motivated by our SD retrieval problem, we define *targets* as *data files (CSV, spreadsheet, etc.)*. This can easily generalize **to *any* definition of target**. Also, we aim for an efficient method, feasible on a single machine, as opposed to SE-scale parallel infrastructure.

Focused crawlers have been studied in prior work [10, 20, 26, 28, 39]. Some aim to select some websites among thousands [39], as opposed to SDs within a given website; others aim at webpages on certain predefined topics [10, 20, 26, 28]. As we will show, traditional focused crawlers make *assumptions that may not hold in our setting, and which we do not need*, e.g., that all pages on a topic tend to be close within a website [17]. Also, their page utility estimation leads them to perform poorly on our problem (see Sec. 4.5).

Motivated by the large numbers of open, valuable SDs accessible via navigation, we address the challenging problem of **retrieving**

SDs in a website as efficiently as possible, without relying on prior knowledge of the website's structure or content.

Contributions. Our contributions are as follows. (1) We formalize our **graph crawling problem** and show that optimally solving it is intractable (Sec. 2). (2) We propose a **novel approach based on reinforcement learning (RL)** in Sec. 3. Our approach relies on two crucial hypotheses: (i) *links found on similar tag paths within HTML web pages* (a tag path [22, 40] goes from the page root, to a hyperlink tag such as <a>) *lead to similar content*, (ii) *we can learn, from the tag path structure, which tag paths are likely to lead to links towards targets.* (3) We demonstrate the effectiveness and efficiency of our approach through **extensive experiments on diverse websites**, totalling 22.2 millions pages (Sec. 4). Our crawler **outperforms all baselines**, including the closest existing focused crawlers [20]. On some websites we experimented on, in particular very large ones, **our crawler retrieves 90% of the targets accessing only 20% of the webpages** (Sec. 4.1).

We discuss related work in Sec. 5. For space reasons, a proof, extra experiments, and further details are in an extended version in [24], which also provides code to reproduce the experiments.

2 Problem Statement and Modeling

We formalize our SD acquisition as a *graph crawling problem*. We show that a relaxed version thereof is NP-hard, even if the website is known before the crawl (it is not!) in Sec. 2.1. Then we detail the mapping of our problem into the graph crawling framework, including a crucial choice of labels for the graph edges, in Sec. 2.2.

2.1 Graph Crawling Problem

We model a website as a rooted, node-weighted, edge-labeled directed graph. Each node represents a webpage, and each edge is a hypertext link leading from one to another. We fix a countable set $\mathscr{L}$ of labels, e.g., finite sequences of character strings.

Definition 1. *A* website graph *is a tuple* $G = (V, E, r, \omega, \lambda)$*, with:* V *a finite set of* nodes *(representing webpages);* $E \subseteq V^2$ *a set of* edges *(representing hyperlinks);* $r \in V$ *the* root *of the graph (the input webpage);* $\omega : V \to \mathbb{R}^+$ *a* cost *function assigning a positive weight to every node (the cost of retrieving that page);* $\lambda : E \to \mathscr{L}$ *a* labeling *function, assigning a label to each link found in a page.*

On such a graph, we define a *crawl* and its *cost* as follows:

Definition 2. *A* crawl *of a website graph* G *is an* r*-rooted subtree* $T = (V', E')$ *of* (V, E)*. Its total cost, denoted* $\omega(T)$*, is defined as* $\sum_{u \in V'} \omega(u)$.

The graph crawling problem is formalized as follows:

Problem 3. *Given a website graph* $G = (V, E, R, \omega, \lambda)$ *and a subset* V^* *of* targets *in* V *is to find a crawl* $T = (V', E')$ *of* G *with* $V^* \subseteq V'$*, of minimal total cost.*

We define the *frontier* of a crawl $\{u \in V \setminus V' \mid (v, u) \in E, v \in V'\}$: these are nodes that have not been crawled but are pointed to from nodes that have been. Figure 1 shows a website graph as well as a possible crawl and its frontier. Even assuming the graph is fully known in advance, the graph crawling problem is intractable:

Proposition 4. *Given a website graph* $G = (V, E, r, \omega, \lambda)$*, a subset* $V^* \subseteq V$ *and some* $B \in \mathbb{R}^+$*, determining whether there exists a crawl* $T = (V', E')$ *of* G *such that* $V^* \subseteq V'$ *and* $\omega(T) \leq B$ *is NP-complete; hardness holds even when* ω *is a constant function. This holds even when nodes in* V^* *do not have any outgoing links in* G.

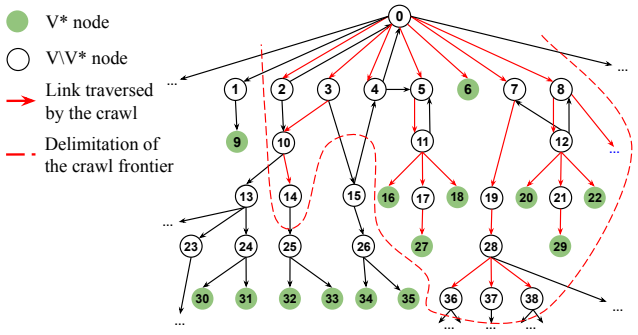


Figure 1: Sample website, crawl, and frontier

This is the decision variant of the graph crawling problem. Hardness is shown by reduction from the set cover problem [23]; the proof can be found in the extended version in [24].

As a consequence, optimal methods being out of reach (interesting websites may have millions of pages), we need **heuristic methods with a low cost in practice**.

2.2 Data Acquisition as Graph Crawling

We complete our modeling of Web data acquisition as an instance of the graph crawling problem by detailing the graph G, the target subset V^*, and the edge labeling function λ. The choice of λ will turn out to be crucial for the performance of our approach.

We identify pages by their URL and fix r, the **website root**, as the start of the crawl. Next, we need to identify which **pages** belong to the website graph. Lacking a commonly agreed definition of a website's boundary [1, 47], we use a pragmatic approach. A URL is considered to be part of the same website as the root r if and only if *its hostname* (the part of the URL which is after the scheme but before the path, with a potential "www." prefix excluded) *is a subdomain of the hostname of* r (a potential "www." prefix also excluded). V contains all such URLs. For instance, if r is https://www.A.B.com/index.php, the URLs https://www.A.B.com/folder/content.php and https://www.C.A.B.com/page.html are part of V, but https://www.B.com/page.php and https://edbticdt2026.github.io/?contents=EDBT_CFP.html are not.[1] An edge $(u, v) \in E$ exists if u links to v via HTML tags like `<a href="...">`, `<area>`, `<iframe>`.

Since our goal is to find SDs, **target webpages** are those whose *Multipurpose Internet Mail Extensions* (MIME) types are in a *user-defined* list (e.g., `application/csv`, `text/csv`, `application/pdf`, `application/vnd.ms-excel`). Non-target types include `text/html`, `video/*`, `audio/*`, `image/*`, etc. The MIME type can be obtained via HTTP HEAD requests; to generalize to other content than SDs, any other target MIME type set can be used. The full list of MIME types we use for to identify targets can be found in [24].

To each **edge** e, the edge labeling function λ associates as label $\lambda(e)$ the **tag path** derived from the HTML page's DOM structure [54], the standard tree-based representation of an HTML page.

[1] The reason for the special handling of a "www." prefix is that many (but not all...) websites use it as a prefix for the domain name of the Web server.

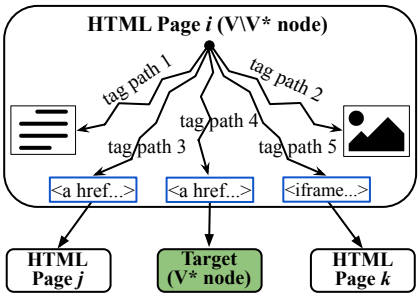


Figure 2: Tag paths in an HTML page

The tag path includes the **full path of HTML tags from the root (the `html` tag) to the hyperlink tag** (`a`, `area`, `iframe`, etc.), along with class and id attributes describing additional structural and styling information. For example, a label might be "`html body div#main ul.datasets li a`", where '#' prefixes the HTML ID, and '.' indicates a class. This labeling makes it very likely that *links labeled with similar tag paths lead to similar content types*, even across different webpages of the same website. Figure 2 zooms within an HTML page. It shows five tag paths (the tags are omitted for readability), leading to: a text paragraph, an image and 3 hyperlinks to other pages (1 target, and 2 other HTML pages). Our labeling function λ captures exactly the tag paths leading to each hyperlink.

We consider two **cost functions** ω: (i) counting HTTP requests, $\omega(u)$=1 for all $u \in V$; (ii) measuring the exchanged data volume, especially data volume the crawler receives with $\omega(u)$ as the page size, which is, in theory, unbounded. We also account for the cost c of *determining if a vertex is in* V^*, typically via an HTTP HEAD request. If ω counts requests, then $c(u)$=1; if ω measures volume, $c(u)$ is based on HEAD response size, usually much smaller than $\omega(u)$. Sec. 3.3 discusses an alternative MIME type prediction method, resulting in a small, amortized c cost, which we can view as constant.

3 Crawling based on Reinforcement Learning

The goal of our efficient crawling is to retrieve as many targets as possible, at a minimum cost. Recall that we assume no prior knowledge of the website's size or structure, thus no knowledge of where the targets are. We hypothesize that *an edge's label* λ (its tag path in the page where the hyperlink appears) *can be used to learn which edges leads to target-rich pages*, and which edges do not. To obtain then leverage such insights, we need to both *explore* the website (to find promising paths), and *exploit* the knowledge thus gained (to retrieve targets). Thus, our approach to focused crawling uses *reinforcement learning* (RL, in short) [49]. Below, we present the environment (states and actions) of our crawling task (Sec. 3.1), and the learning algorithm (also called the *learning agent*) that evolves in this environment (Sec. 3.2). We describe the URL classifier we use to compute the agent's rewards in Sec. 3.3, and the overall crawling algorithm in Sec. 3.4.

3.1 Environment: States and Actions

In RL, an *agent* evolves in an *environment*. It starts from an initial *state*, where it can choose among a set of *actions*. Each *action* leads to a new *state*, where other actions can be chosen. Each choice leads to a *reward*. The goal of the agent is to learn a *policy* (a mapping from states to actions), that maximizes the total reward. This model is known as a *Markov Decision Process* (MDP) [27].

Let us consider what we could use as *state* in our crawler. As the crawl starts from the root and visits other pages, one could think of using "the currently crawled webpage" as state. However, this choice is not suitable: the current webpage does not reflect previous choices (the webpage may be accessible via several navigation paths). Also, an efficient crawler should not visit any webpage twice; in contrast, learning supposes multiple visits to a state, to refine and then leverage information learned in that state.

To keep the model simple and lightweight (see Sec. 5 for discussion), we use a **single-state model**, where the agent is perpetually in exactly only one state. What matters then is only the set of *actions* that the agent can follow at each crawl step. Intuitively, an action is to navigate along a link. We delve into action details below.

3.2 Grouping Links into Actions

Following our hypothesis that *links with similar labels (tag paths) have similar values*, i.e., likelihood of leading to targets, we model an action as *a group of similar links*, with the semantics that taking the action amounts to crawling along one of these links.

Since each link is labeled with its tag path in its enclosing page, we need a **measure of similarity between tag paths**. To that effect, we represent each tag path by a *numerical bag-of-words (BoW) vector* p, over the n-grams vocabulary built from all tag paths encountered so far. Note that n-grams preserve the order of HTML node names appearing in tag paths; in our context, this order is significant (as Sec. 4 confirms). Since the n-grams vocabulary is *dynamically* built during the crawl, BoW vectors assigned to tag paths encountered at different times have different lengths. To be able to compute distances between these representations, we first *project each tag path into a fixed-dimensional vector* of size $D = 2^m$ for a chosen $m > 0$. We do this as follows:

(1) We fix a *hash function* $h : x \mapsto \left\lfloor \frac{(\Pi \times x) \mod 2^w}{2^{w-m}} \right\rfloor$ which maps any integer x to a number between 0 and $D-1$. Its parameters Π (a large prime number) and w are fixed, and chosen such that $w > m$.

(2) Calling h on each position between 0 and $d-1$, where d is the length of the BoW vector p, maps it to a new position between 0 and $D-1$. This enables us to transform p into a D-dimensional p_D vector, where for every $0 \leq i < d$, $p_D[h(i)] = p[i]$.

(3) Potential collisions of h, i.e., $i_1 \neq i_2$ such that $h(i_1) = h(i_2)$, are resolved by assigning to $p_D[h(i_1) = h(i_2)]$ the *mean* of all elements of p, at positions which collide with i_1, i_2. Further, there may be positions j between 0 and D such that for every $0 \leq i < d$, $h(i) \neq j$. For such positions j, $p_D[j] = 0$. This happens, e.g., at the beginning of the crawl, when the set of HTML element names seen so far is small (thus d is small).

Figure 3 illustrates tag path projection for an example where $D = 2^m = 4$, $w = 11$ and $\Pi = 766\,245\,317$. At iteration k, the 2-gram vocabulary contains $d_k = 5$ elements (including special tokens `BOS` and `EOS` denoting beginning and end of stream). At iteration $k+1$, a new tag path is vectorized into a 2-gram vector of dimension 10, and the vocabulary is updated accordingly, totaling in this case $d_{k+1} = 11$ elements. From it, we compute the d_{k+1}-dimensional BoW vector p_{k+1}. The hash function h then maps each input dimension $i \in \{0, \ldots, d_{k+1}-1\}$ to $\{0, \ldots, D-1\}$. For example,

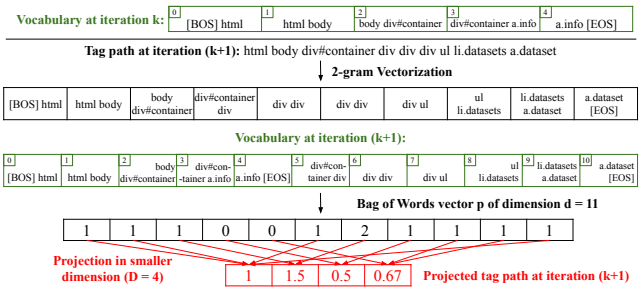


Figure 3: Mapping of a tag path into a fixed-size vector.

$h(2) = \left\lfloor \frac{(766\,245\,317 \times 2) \mod 2048}{512} \right\rfloor = 1$. Collisions are likely to occur: for instance $h(4) = h(8) = h(9) = 3$, so we average: $p_{Dk+1}[3] = \frac{p_{k+1}[4]+p_{k+1}[8]+p_{k+1}[9]}{3} \approx 0.67$. We finally get a four-dimensional p_{Dk+1}.

Our next task is to **find, for each projected tag path representation p_D, the nearest action a_c** among all known actions A. Conceptually, we see an action as an **evolving set (cluster)** of similar tag paths with a **centroid**, which is the mean of all projected tag paths. For efficiency, for each action, we only store the centroid, which changes as the action's set of tag paths evolves.

Algorithm 1 computes the nearest action of each p_D if it is close enough; otherwise, a new action is created and returned.

For that, we insert the action centroids in a *Hierarchical Navigable Small Worlds* (HNSW) [38] index denoted $\mathcal{I}$, chosen for its highly efficient updates when a new tag path joins an action (requiring the update of its centroid). While maintaining and querying such an index might be costly in CPU time, this cost is negligible compared to the crawl time (waiting between two requests, or for the download of webpages). The index also gives an estimation of the *cosine similarity* between p_D and its closest centroid. If the similarity is above a **threshold** θ, the tag path is added to the action. Otherwise, a new action composed of only p_D is created. As our experiments show (Sec. 4), the choice of the threshold has a significant impact on the performance of our learning agent. As an extreme case, if $\theta = 0$, all tag paths are grouped into a single action, and the agent cannot learn anything: it will randomly select the tag path to follow. On the other hand, if $\theta = 1$, each tag path is a separate action. Such an agent only explores, and never exploits, since no action can be repeated. A useful threshold should enable enough actions to separate interesting groups of tag paths from non-interesting ones, while still allowing learning and exploitation.

The family of reinforcement models dedicated to single-state environments is called *Multi-Armed Bandits* (MABs) [49]. The standard, deterministic, MAB algorithm optimizing the trade-off between exploration and exploitation is *Upper-Confidence Bound* (UCB) [3]. It is based on the principle of *optimism under the face of uncertainty*: at each time step, a score is computed for each action, and the action with the highest score is chosen.

The MAB score assigned to a possible action a at time $t+1$, denoted $s^{t+1}_{\text{MAB}}(a)$, includes an **exploitation term** (the mean of rewards obtained so far for that action), and an **exploration** term, reflecting the number of times the action has been selected relative

Algorithm 1: Finding the action for a hyperlink l, $\lambda(l)=p$

Input: Action set A, projected tag path p_D
Output: Action a
$a_c \leftarrow$ Approx. nearest neighbor (from index $\mathcal{I}$) of p_D;
$s_c \leftarrow$ Cosine similarity between a_c and p_D;
if $s_c \geq \theta$ **then**
  Update centroid of a_c in $\mathcal{I}$ with respect to p_D;
else
  Add a new entry a_{new} to $\mathcal{I}$, with value p_D;
  Add a_{new} to A;
  $a_c \leftarrow a_{\text{new}}$;
return a_c

to the crawling history length: $s^{t+1}_{\text{MAB}}(a) = R^t_a + \alpha\sqrt{\frac{\log(t)}{N_t(a)+\epsilon}}$, where R^t_a is the mean reward obtained for action a until time t, α is a parameter weighing exploration vs. exploitation, $N_t(a)$ is the number of times action a has been selected up to time t, and $\epsilon > 0$ is a small value to to avoid division by 0 when $N_t(a) = 0$.

Since each URL is visited only once, an action may become **empty** after all its URLs are visited. The *Awake Upper-Estimated Reward* (AUER) [33] is an alternative to UCB, which shows how to compute the reward when some actions become unavailable, or *sleeping*. Following the AUER model, our score becomes: $s^{t+1}_{\text{SB}}(a) = \mathbb{1}_a(t) \times \left(R^t_a + \alpha\sqrt{\frac{\log(t)}{N_t(a)+\epsilon}}\right)$ where $\mathbb{1}_a(t)$ is 1 if there are remaining unvisited links in action a at time-step t, and 0 otherwise. After selecting an action a, our crawler *randomly chooses an unvisited link l from a to traverse next*, with an equal selection probability for all links within a. How should we reward the crawling along $l \in a$? Considering that our goal is to find *new targets*, the reward should reflect the number of *not-yet-discovered links to targets*, contained in the HTML page h reached via l. This "novelty" condition is set to help the learning agent focus on unseen targets. For instance, if h contains 12 links, including 5 leading to targets, with 2 of these already retrieved, the reward for following l to h should be 3. This simple choice gives good results, as shown in Sec. 4.

Yet, a challenge remains: we must **compute the reward of all links in h before following any of them**. Since following a link incurs a crawling cost, a (fast) *estimation* is needed. Additionally, we need this estimation for *any* link, even though (depending on how the crawl evolves) some links in h may actually *never* be followed. For this, we introduce a *URL classifier*, which, just by inspecting the links in the newly acquired page h, can compute the reward. This classifier is *online*, meaning it evolves during the crawl. We detail it in Sec. 3.3.

Last, we discuss the value of the parameter α in our learning agent. UCB and AUER are known to be optimal when $\alpha = 2\sqrt{2}$ [3, 33]. However, this holds only under standard conditions where the rewards are normally distributed in $[0, 1]$. In our case, rewards, which count new targets, are unbounded. Further, they typically do not follow a normal distribution (see Sec. 4.7). In fact, we *hope* this is *not* the case: it would mean that a majority of HTML pages contain links to targets. If this held, we would have to visit the entire website, or at least a vast share, to retrieve most targets. Setting up α to guarantee optimality in our case would require

knowing the shape of the distribution of rewards, which is not possible. Therefore, we keep $\alpha = 2\sqrt{2}$, which gives good results in practice, across very varied reward distributions (as shown in Sec. 4.1, in particular in Figure 4).

3.3 Estimating Rewards with URL Classifier

Determining the reward of an action taken by our agent (crawl a link) requires assessing a page's MIME type from its URL. One way to do so is by making an **HTTP HEAD** request to the website. Such remote calls are still quite expensive, and they require spacing, e.g., by 1 second, for crawling ethics. Using the **extension** (last URL segment), e.g., `.html`; `.pdf`, `.csv`, etc., fails for links without extensions, such as https://www.justice.gouv.fr/en/node/9961, or others within ILO[2]. To be efficient and generic, we devise an **online URL classifier**[3] **to estimate rewards**, as follows.

For our purposes, any URL belongs to one of **three classes**: "HTML", "Target" or "Neither". An HTML page is added to the crawl frontier and possibly eventually crawled, while a target adds to the crawler's reward. The "Neither" class includes URLs of pages that are not HTML and whose MIME type is not a target one, as well as URLs lacking a MIME type from the server. Among this "Neither" class, in our experience, more than 92% cases correspond to HTTP **4xx** or **5xx** codes, designating errors on client or server.

We initially set our classifier to assign one of these three classes to each link found in a newly crawled page h. However, it proved unable to distinguish URLs resulting in 4xx or 5xx errors, from those that resolve fine. Intuitively, this is because the former URLs are often very similar to URLs of "HTML" or "Target" pages that are accessible. However, *different classification errors do not have the same impact on crawling*: (1) Misclassifying a "Neither" URL as "HTML" or "Target" only incurs the extra cost of following the URL (later, respectively, immediately); the cost is moderate, since it will likely throw an error. To further reduce visits to non-error pages, we include filters based on user-defined MIME types and URL extensions (see Sec. 3.4). (2) Misclassifying "HTML" or "Target" as "Neither" *completely withdraws the page from the crawl*: if it is a target, we miss it; if it is HTML, we miss all URLs that might be discovered by following it. This can make the crawler miss huge parts of the website. To avoid the second kind of errors, our classifier is trained on just **two classes ("HTML" and "Target")**, even though we know some URLs are neither.

Algorithm 2 details the training and usage of our classifier $\mathscr{C}$. It is a logistic regression model [31], iteratively trained through epochs of *Stochastic Gradient Descent* (SGD) [8], with batch size b. The classifier takes as input a vector of character-wise 2-grams of the original URL. For instance, the URL https://www.A.com/data/file.csv is transformed into a list [ht, tt, tp, … , .c, cs, sv]. We associate to each pair of usual ASCII characters[4] a unique identifier, and encode each URL as a bag-of-words over this vocabulary.

As Algorithm 2 shows, during the first training epoch, we label b URLs using HTTP HEAD requests, with X representing the set of URLs, and y their labels. Based on this, we compute rewards and assign URLs to either the frontier $\mathscr{F}$ or the set of targets V^*. After this epoch, we set the Boolean initial_training_phase to false, and use the classifier to infer URL classes without additional HTTP HEAD requests. The classifier is further improved through *online training*: **during execution, any HTTP GET request issued by our crawl contributes an annotated (URL, class) pair**, which we add to our classifier's X and y for incremental training, when reaching the batch size. In this way, after the first training epoch, we get labels at no extra cost.

Algorithm 2: Online URL classifier procedure

Input: U, a URL
Output: $l_U \in$ {"HTML", "Target"}
if $|X| \geq b$ **then**
  $X_{\text{mat}} \leftarrow$ 2-gram bag-of-words of each URL in X;
  $\mathscr{C}$ is incrementally trained on batch (X_{mat}, y);
  $(X, y) \leftarrow (\emptyset, \emptyset)$;
  initial_training_phase $\leftarrow$ False;
if initial_training_phase **then**
  $l_U \leftarrow$ MIME type class from HTTP HEAD on U;
  Add (U, l_U) to (X, y);
  return l_U
else
  $U_{\text{vec}} \leftarrow$ 2-gram bag-of-words vector of U;
  return $\mathscr{C}(U_{\text{vec}})$

Since we want to both *use* the classifier as soon as possible and *train* it often to improve its accuracy, we must set b relatively small. This online method **adapts to potential changes in the form of the URLs**, e.g., when the crawl discovers new parts of the website where URLs are formatted differently. An off-line method would not be able to adapt, even if trained on many examples initially.

3.4 Crawling Algorithm

We now present the overall crawling procedure, described in Algorithm 3. Initially, the tree T of already crawled pages, the set A of actions and the frontier $\mathscr{F}$ are empty. The **budget** β spent by the crawler up to this point, and the crawling step (or time) t are both set to 0. The crawler starts by visiting the original link r, continuing until either all links are visited (the website is completely crawled) or the maximum budget B is reached. At each step, if the set of actions is not empty, the learning agent chooses an action a_c following the Sleeping-Bandit (SB) procedure of Sec. 3.2. Among the links in $\mathscr{F}$ associated with action a_c, the agent choses a link uniformly at random. Then, it crawls the page behind this link with Algorithm 4. If the MIME type of the page matches a predefined blocklist during download, its retrieval is immediately interrupted; this is typically used to avoid downloading multimedia content.

Algorithm 4 starts by retrieving some information from the HTTP GET response: the HTTP status, the MIME type from the header, and the body (content) of the webpage. The cost of the request is added to the budget β. The HTTP response status may fall into one of three categories: (1) Errors (on the client or server side) marked by **4xx** or **5xx** statuses, do not yield new links or targets. (2) Redirections, indicated by **3xx** statuses, include an extra "Location" header with the redirection URL. If the link hasn't been crawled yet, the crawler follows and processes it. (3) A **2xx** status

[2]E.g., https://www.ilo.org/publications/elevating-potential-rural-youth-paths-decent-jobs-and-sustainable-futures-1.
[3]Observe that here we classify *URLs*, whereas in the previous section we clustered *edge labels*, which, crucially to our method, are *tag paths* within HTML pages (Fig. 2).
[4]ASCII digits, upper and lower case letters and main special characters.

Algorithm 3: Efficient target retrieval

Input: r the initial page to crawl
$A, \mathscr{F}, T \leftarrow \emptyset$; $\beta, t \leftarrow 0$; $u \leftarrow r$; Add r to $\mathscr{F}$;
while $|\mathscr{F}| > 0$ *and* $\beta \leq B$ **do**
  if $|A| > 0$ **then**
    $a_c \leftarrow \arg\max_{a \in A} \mathbb{1}_a(t) \left(R_{\text{mean}}(a) + \alpha \sqrt{\frac{\log(t)}{N_t(a)+\epsilon}} \right)$;
    $u \leftarrow$ Select a link from $\mathscr{F}$ mapped with action a_c uniformly at random, and $N_t(a_c) \leftarrow N_t(a_c) + 1$;
  else
    $u \leftarrow$ Select a link from $\mathscr{F}$ uniformly at random;
  crawl_next_page(u) (Algorithm 4);

Algorithm 4: Crawl next page

Input: u the URL to be crawled
Add URL u to T, and $t \leftarrow t + 1$;
http_response, mime_type, body $\leftarrow$ Result of the request on URL u with HTTP GET, and update β accordingly;
if request interrupted (banned MIME type) **then return**;
if http_response *is 4xx or 5xx (error)* **then return**;
else if http_response *is 2xx (success)* **then**
  if mime_type $\neq \emptyset$ **then**
    if *"*HTML*"* $\subset$ mime_type **then**
      Add (u, "HTML") to (X, y);
      $U_{\text{new}} \leftarrow$ All hyperlinks in body pointing to the same website as r;
    else if mime_type $\in L$ **then**
      Add (u, "Target") to (X, y);
      Add target (the content of body) to V^*;
    return;
else if http_response *is 3xx (redirection)* **then**
  $u \leftarrow$ Location from HTTP GET result on u;
  if $u \notin T \cup \mathscr{F}$ **then** crawl_next_page(u) and **return**;
reward $\leftarrow 0$;
for $u_{\text{new}} \in U_{\text{new}}$ such that $u_{\text{new}} \notin T \cup \mathscr{F}$ **do**
  if has_blocklisted_extension$\left(l_{u_{\text{new}}}\right)$ **then continue**;
  $l_{u_{\text{new}}} \leftarrow$ class_of_url(u_{new}) (Algorithm 2);
  Update β if HTTP HEAD request was made;
  if $l_{u_{\text{new}}}$ = *"HTML"* **then**
    $a_c \leftarrow$ map_link_to_action(A, u_{new}) (Algorithm 1);
    Add u_{new} to $\mathscr{F}$;
  else if $l_{u_{\text{new}}}$ = *"Target"* **then**
    crawl_next_page(u_{new}), and reward $\leftarrow$ reward + 1;
if $u \neq r$ **then** $R_{\text{mean}}(a_c) \leftarrow R_{\text{mean}}(a_c) + \frac{\text{reward} - R_{\text{mean}}(a_c)}{N_t(a_c)}$;

means the server successfully responds with either an HTML page, a target (that is, a page whose MIME type is in the list L presented in Sec. 2.2), or neither. In the first two cases, X and y are updated. For HTML pages, new hyperlinks U_{new} are extracted from the page, and we keep only the links within the same website. For each link, if the corresponding URL is not already in $\mathscr{F}$ or in T, it is assigned a class by calling Algorithm 2, except when its extension matches a predefined blocklist established before the crawl.

If it points to an HTML webpage, we map the link to an action using Algorithm 1. The set of actions A is updated, either by *changing an action's centroid*, or by *adding a new action*, and the chosen action a_c is mapped to the link, which is then added to frontier $\mathscr{F}$. If the link leads to a target, we immediately retrieve it by recursively calling Algorithm 4, and the reward is updated. Finally, if the page crawled by the current call of Algorithm 4 is not the starting page, the mean reward for the chosen action a_c is updated to reflect the last computed reward.

4 Experimental Results

4.1 Websites

Table 1 lists the 18 websites used in our experiments. As our primary application is a collaboration with French journalists [7] from RadioFrance, six websites are French, public, and governmental: the French National Assembly (*as*), French Local Communities (*cl*), French Council for Statistical Information (*cn*), and the ministries of Interior (*in*), Education (*ed*), and Justice (*ju*). Each contains material on government branches, including statistics (our crawler's targets). We also include eight multilingual websites: the UN's International Labor Organization (*il*), French Official Statistical Institute (*is*), Japan's Ministry of Interior (*jp*), OECD (*oe*), Open Knowledge Foundation (*ok*), Qatar's Official Statistical Service (*qa*), the UN World Health Organization (*wh*), and the World Bank (*wo*). Finally, we add national websites in English: the Australian Bureau of Statistics (*ab*), US National Center for Education Statistics (*nc*), the US Census (*ce*), and the US Bureau of Economic Analysis (*be*). Websites with pages in at least two languages have a ✓ in the "**Mlg.**" column. The websites range from a few thousand to over a million pages, excluding those resulting in 4xx or 5xx HTTP errors (see "**#Available (k)**" column). A ✗ in the "**Fully Crawled**" column indicates incomplete crawls (Sec. 4.4).

The following metrics assess the *difficulty* and *interest* of a focused crawl. For partially crawled websites, metrics are computed on the BFS-visited subset (Sec. 4.4). "**#Target (k)**" is the total number of identified targets. The ratio $\frac{\#\text{Target (k)}}{\#\text{Available (k)}}$ measures *target density*; extremes are 66.78% in *cl* and 2.49% in *in*. "**HTML to Target (%)**" is the percentage of HTML pages linking to at least one target, reflecting the *density of target-pointing pages* (e.g., *cl* has the highest density, but all links are concentrated in 5.40% of the HTML pages). "**Target Size (MB)**" is the average target file size and its standard deviation (STD). "**Target Depth**" gives the mean and STD of target depths, defined as the shortest link navigation from the root. *Mean depths vary greatly, from below 3 to nearly 90.* Highest values arise on portals requiring page-to-page navigation, e.g. *ju*. Table 1 shows our websites are *extremely diverse*: small or huge, with varying depths, numbers of targets, target locations, and over 20 languages. Such diversity demands *very different crawling efforts*, requiring the crawler to adapt to quickly retrieve targets in each of them. A manual analysis of typical target locations on *ju*, *il*, *wh*, and *nc* is provided in the extended version in [24].

4.2 Search Engines and Dataset Search

We attempted to retrieve SDs via search engines (SEs), in particular, three popular Google services: classical search (GS), Google

Table 1: Main characteristics of websites (see detailed crawling methodology in Sec. 4.4)

	Starting URL	Mlg.	Fully Crawled	#Available (k)	#Target (k)	HTML to Target (%)	Target Size (MB)	Target Depth
ab	https://www.abs.gov.au/	✗	✗	952.26	263.26	8.86	4.50 (± 56.04)	8.94 (± 2.56)
as	https://www.assemblee-nationale.fr/	✗	✗	949.42	155.94	4.34	0.54 (± 6.38)	5.84 (± 1.07)
be	https://www.bea.gov/	✗	✓	31.23	15.84	32.19	2.03 (± 6.99)	5.73 (± 3.21)
ce	https://www.census.gov/	✗	✗	988.37	257.68	3.47	1.51 (± 15.77)	4.23 (± 0.48)
cl	https://www.collectivites-locales.gouv.fr/	✗	✓	5.54	3.70	5.40	1.15 (± 4.91)	2.80 (± 0.82)
cn	https://www.cnis.fr/	✗	✓	12.80	7.49	13.87	0.43 (± 1.74)	4.26 (± 1.59)
ed	https://www.education.gouv.fr/	✗	✓	102.71	10.47	3.95	1.00 (± 3.07)	11.89 (±13.22)
il	https://www.ilo.org/	✓	✗	990.71	81.01	2.53	13.40 (±110.01)	4.26 (± 1.28)
in	https://www.interieur.gouv.fr/	✗	✓	922.46	22.98	1.54	1.12 (± 3.06)	66.94 (±39.43)
is	https://www.insee.fr/	✓	✓	285.55	168.88	41.34	3.13 (± 21.43)	5.20 (± 1.81)
jp	https://www.soumu.go.jp/	✓	✗	993.87	328.83	6.30	0.80 (± 4.49)	5.18 (± 1.29)
ju	https://www.justice.gouv.fr/	✗	✓	56.61	14.85	4.85	0.48 (± 1.34)	86.91 (±86.30)
nc	https://nces.ed.gov/	✗	✓	309.97	84.94	18.87	1.10 (± 11.56)	3.63 (± 1.66)
oe	https://www.oecd.org/	✓	✓	222.58	45.04	15.61	2.31 (± 23.37)	6.28 (± 5.65)
ok	https://okfn.org/	✓	✓	423.12	12.95	0.74	0.04 (± 0.24)	2.64 (± 2.89)
qa	https://www.psa.gov.qa/	✓	✓	4.36	2.45	4.15	2.97 (± 19.28)	3.03 (± 0.61)
wh	https://www.who.int/	✓	✗	351.86	55.59	14.19	1.26 (± 11.14)	4.43 (± 0.62)
wo	https://www.worldbank.org/	✓	✗	223.67	23.10	2.38	2.80 (± 27.16)	4.52 (± 0.69)

Datasets Search (GDS), and the Google Public Data Explorer (GPDE) providing open data from international organizations.

GS allows filtering by website (site:) and file type (filetype:), through both its Web interface and API. However, SEs, including Google, **do not fully index all websites**, and GS **limits results to 1k**, often truncating them. For instance, querying *ju* for PDFs yields only 302 results, although more than 9k exist. Similar gaps appear for other sites and MIME types: on *ju*, we only get 240 ODS files (out of 910), and on *in* 38 XLS (out of 1 546). Even worse: TSV is not recognized at all despite 11 097 files on *ju*. On *il*, GS lists 641 results instead of over 49k, and 0 ZIP archives instead of at least 2 239.

GDS **trims results even further**, e.g., only 109 tabular files for *ju* (out of 1 188); on *il*, 93 datasets (vs. at least 170k found by our crawler); and on *ce*, 312 datasets (vs. 800k), etc.

GPDE is built for human readers, indexing from a **closed, fixed set of providers**, such as Eurostat, *wo*, Statistics Canada, and *oe*, but excludes key sites like UN's *wh*, *il*, and US agencies (*be*, *nc*, *ce*). **GPDE only allows manual exploration**, plotting datasets guided by users' filters but **offering neither downloads nor links to originals**: e.g., it displays 150 OECD statistics across 37 countries and 100 years, but only references a 272-page PDF (OECD iLibrary).

Overall, SEs lack **transparency and control over the retrieved SDs**, a critical limitation for our task. In contrast, our crawler retrieves all data files within a specified budget, with explainable decisions and explicit choice of target MIME types.

4.3 Baselines

First, we consider four simple baselines. (i) The RANDOM crawler selects the next hyperlink to visit uniformly at random from the crawl frontier. (ii) The *Breadth-First Search* (BFS) exhaustive crawler maintains the frontier as a *First-In-First-Out queue* data structure. It crawls all pages reachable by a path of length ℓ before any page reachable only by longer paths ($\ell' > \ell$). (iii) *Depth-First Search* (DFS) maintains the frontier in a *Last-In-First-Out stack*. It is rarely used for exhaustive crawling since it may fall into robot traps. (iv) The unrealistic *Omniscient Crawler* (SB-ORACLE) knows all target URLs (V^*) from the start and crawls them sequentially. Since an optimal crawler is intractable (Prop. 4), this omniscient crawler provides an (unreachable) upper bound on crawler efficiency.

The *Offline-Trained, Tag Path-Based Crawler* (TP-OFF) [20] is a close competitor from the literature, designed to retrieve *diverse* textual content. It first crawls 3 000 pages with BFS and groups the tag paths leading to followed links as in Sec. 3.1. Page benefits are computed immediately, and tag path groups are stored in a priority queue ordered by average benefit. After these 3 000 pages, [20] *only considers links matching existing tag path groups* (ordered by priority), other are ignored. In our context, benefit computation requires knowing if links lead to targets. To enable this baseline, *we provide it with the true benefit* (number of targets behind a page's links) as if given by an oracle, for the initial 3 000 pages. Beyond this point, new tag path groups can form but receive a permanent benefit of zero. This baseline, given an unfair advantage in its first stage, can be seen as an *ablated* version of our crawler, learning *offline* instead of *online* with our RL method.

The *Focused Crawler* (FOCUSED) uses a classifier to assess the likelihood that a new hyperlink leads to a target. The frontier is maintained as a *priority queue*, hopefully favoring targets. It employs a standard *logistic regression*, periodically retrained on crawled pages, thus at no extra HTTP cost. Its features follow standard focused crawler practice as in, e.g., [10, 19]: the (approximated) depth of the page where the link was found, the 2-gram vector representation of the URL (similarly to Sec. 3.3), and the 2-gram vector representation of its *anchor* text. *Topic-oriented* features are excluded (further discussed in Sec. 5). This baseline can be seen as an *ablated* version of our crawler, which neither benefits from tag path structure information (Sec. 3.3), nor from RL (Sec. 3.4).

TRES [36] is a topical, RL-based focused crawler that uses a Bi-LSTM classifier to assess the relevance of HTML pages. Like our approach, TRES uses RL to learn where to find interesting content in a website. However, since TRES targets text within HTML pages only, it requires in input keywords and examples of relevant pages, and only applies to one language. To include TRES as a baseline, extensive adaptations were required. Specifically, we gave it three *unfair advantages* not given to other crawlers: (i) TRES **alone** needs a list of **topic-specific keywords** to initialize its classifier's training. We supplied 74 hand-crafted terms likely to appear in links' anchors to targets (full list in [24]). (ii) Its classifier requires HTML pages with links to targets as positive examples before crawling to be trained. **We manually provided 1 000 such pages** from the 10 evaluation websites (based on prior crawls), giving it partial access to the solution **at no cost**. (iii) TRES must classify URLs as HTML or not to manage its frontier, which only accepts HTML pages. Although costly in practice (see Sec. 3.3), we gave it an **oracle** doing this classification **at no cost**. Because the RL agent and classifier rely only on textual HTML features, TRES **cannot be trained to detect targets directly**. We adapted it as follows: only links to HTML pages are added to the frontier (as in [36]); other links are visited immediately, and targets are counted if found. We also **disabled its language filter**, which excluded non-English pages using coarse URL-based heuristics prone to false exclusions. The modified code is available in [25].

4.4 Practical Crawling in our Experiments

To evaluate six baselines and our crawler with various hyper-parameters, repeated crawls over each website are not feasible due to (a) time constraints (especially with large files and/or slow connections), and (b) crawling ethics. Thus, for each website we run all baselines and our crawler in a setting where *each crawler first checks if the resource is already stored in a local database*. If so, we use it; otherwise, we fetch it via HTTP GET and the result (URL, HTTP status, headers, and response body) is stored in the database.

For evaluation purposes only, we stop crawling on a given website, as follows: (i) If any crawler terminates before reaching **1M** pages, the entire website is crawled, yielding a complete local replication: all other crawlers can then be run just using local database look-ups. (ii) If each crawler visited **1M** webpages (possibly not the same pages across crawlers). (iii) If some crawlers are still running after a **3-week** time limit, as crawling can be extremely slow for some websites: across our 18 test websites, we retrieved **22.2M distinct webpages**. (iv) If a crawler takes more than 1 minute between two requests, excluding network latency (this only affected TRES).

We conduct and present our experiments as follows: (i) We carry **hyper-parameter studies** and **evaluate our URL classifier** on fully-crawled websites with *fewer than 1M pages*. Fully replicated websites allow us to run crawlers in parallel under multiple parameter settings, and to **gather complete knowledge** for both (unrealistic) SB-ORACLE crawler and TRES baseline (see Sec. 4.5). (ii) On websites where *all baselines crawled 1M webpages*, crawlers are compared **on the subset of the website each of them visited**. Note that their target sets, and the volume of retrieved data, may differ: a good crawler prioritizes *more target-rich* pages. (iii) On websites where *at least one baseline did not crawl 1M pages within the time limit*, crawlers are compared **on the smallest number of pages** visited by any crawler (the first crawled pages for those who crawled more): e.g., if three crawlers visit 400k, 200k, and 800k pages, respectively, we compare them on their first 200k pages.

We exclude retrieval times from our results, since these vary out of our control. We focus on the *number of requests and data volumes*; crawl time can be estimated from these, knowing the bandwidth and the wait between two consecutive requests.

4.5 Comparison with Baselines

Figure 4 compares our crawler's performance with the baselines on 10 diverse, representative websites; the other plots are in the extended version [24]. Default settings are $n = 2$ (n-grams), $\theta = 0.75$, $\alpha = 2\sqrt{2}$, $m = 12$ (path vector dimension), $w = 15$ (hash function), and $b = 10$ (batch size). MIME type and extension blocklists are chosen to filter out multimedia content (image, audio, video), as they are usually large in volume. Full lists are provided in [24].

In addition, for each crawler and all 18 websites, Table 2 (above the double rule) shows *the percentage of requests needed to retrieve 90% of targets*, and Table 3 *the percentage of non-target page volume retrieved before reaching 90% of total target volume*. Lower values indicate greater efficiency. For partially crawled sites, metrics are computed on pages visited by BFS (Table 1). If a crawler never reaches the desired 90%, we show $+\infty$. In both tables, the best-performing crawler per website is highlighted, excluding SB-ORACLE, which does not exist in practice. For each of the 10 websites in Figure 4, two graphs show crawler performance: on the left, the *number of crawled targets v.s. number of HTTP requests* (both GET and HEAD, in thousands); on the right, the *volume of target responses (GB) v.s. volume of non-target ones*. In both graphs, *a higher curve is better*.

For *fully-crawled websites*, we run two variants of our crawler: SB-CLASSIFIER, and SB-ORACLE, where we replace the URL classifier is replaced by a (unrealistic) perfect oracle. This quantifies the impact of the URL classifier on crawl performance. Results for both are *averaged over 15 runs*, with shaded areas indicating ± one STD. STD are in most cases low, indicating stable behavior. Variability arises mainly from: randomness in link selection within chosen actions; the stochastic SGD-trained classifier; and the single-state RL agent, whose design is crucial for stabilization (see extended version [24]).

For *partially crawled websites*, we only show a single run of SB-CLASSIFIER, lacking perfect classifier information, and since multiple crawls are unfeasible (Sec. 4.4). All baselines but the random one and TRES are deterministic, thus one run suffices. The TRES baseline is only evaluated on the 10 smallest, fully-crawled websites as it requires oracle access and does not scale. Except for small websites (*be*, *cl*, *cn*, and *qa*), TRES exceeds the threshold of 1 minute per request and is thus stopped before completing the crawl of larger websites. Note that crawling 100k additional pages at that point would require (at the very least) 10 more weeks.

The plots and Tables 2 and 3 show that **our crawler, with the actual URL classifier, outperforms all baselines**, with one exception: on *cl*, DFS is slightly better toward the end. On *il*, on volume only, BFS temporarily outperforms ours, but overall our crawler retrieves twice more targets. **For websites *as, ce, ed, il, in, ju, nc, wh,* and *wo,* our approach significantly reduces resource usage (time or bandwidth)** to reach a fixed number of targets or maximize targets under a budget, demonstrating practical usefulness. For instance, on *wo*, the best baseline (BFS) would take 3

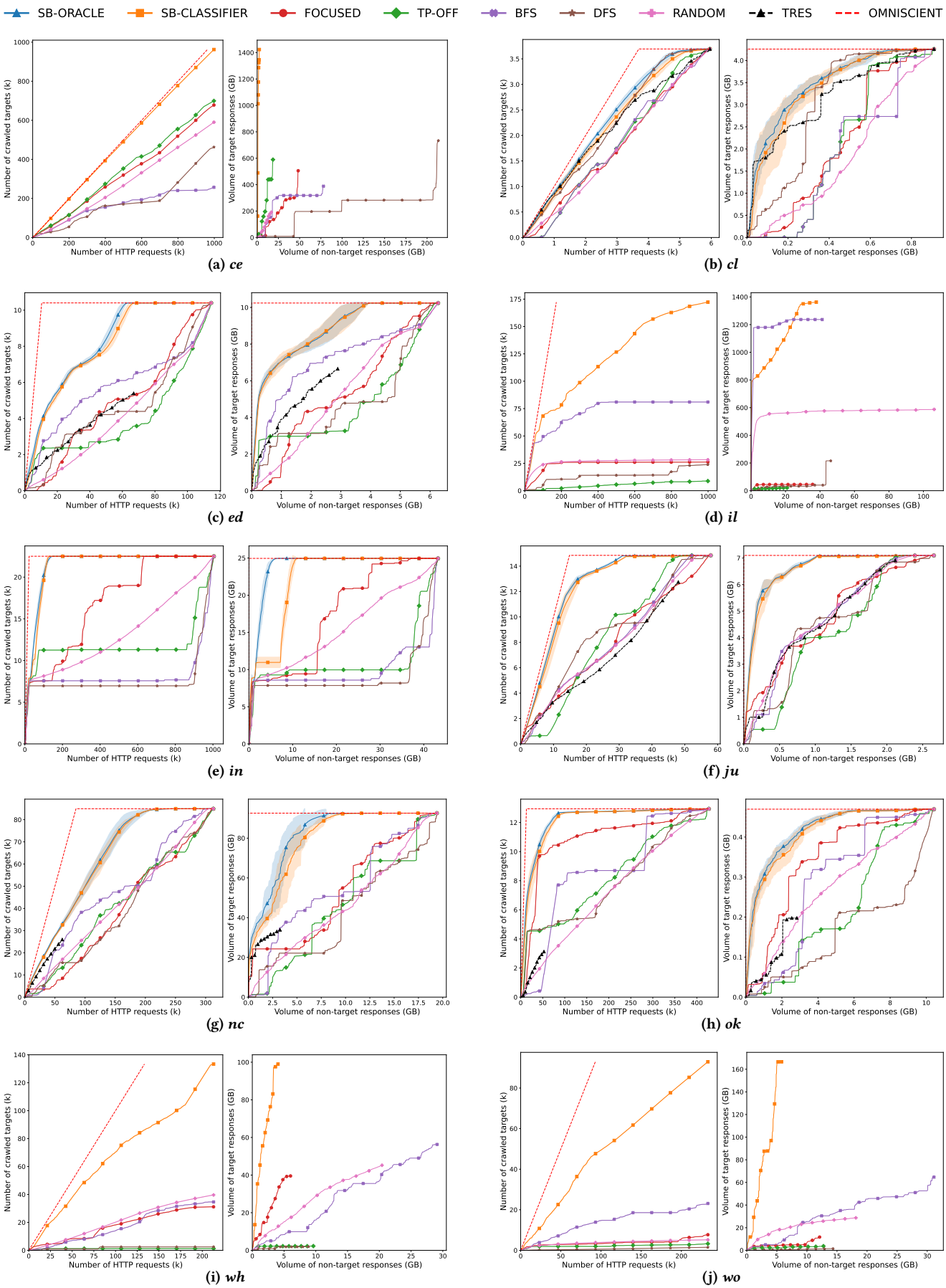


Figure 4: Comparison of different crawler performance for 10 selected websites presented in Table 1; for TRES, experiments are only shown for fully-crawled websites

Table 2: Percentage of requests that each crawler performs to retrieve 90% of the targets, for all websites. Below double horizontal rule, percentage of saved requests and percentage of lost targets due to early-stopping mechanism (see Sec. 4.8).

Crawler	*ab*	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*
SB-ORACLE	NA	NA	72.6	NA	70.7	70.3	48.0	NA	12.8	73.8	NA	34.1	50.8	55.8	13.8	47.3	NA	NA
SB-CLASS.	**31.2**	**35.1**	**75.7**	**23.5**	74.4	**70.9**	**51.5**	**14.2**	**11.9**	**76.0**	**37.7**	**35.8**	**51.6**	**59.2**	**15.5**	**57.7**	**19.7**	**18.6**
FOCUSED	68.2	+∞	87.8	36.0	88.9	82.7	86.7	+∞	62.8	86.9	42.0	91.1	92.8	84.9	51.8	71.0	+∞	+∞
TP-OFF	96.4	50.3	86.2	34.7	81.8	88.2	95.6	+∞	99.7	88.0	+∞	74.4	93.0	88.7	76.2	88.6	+∞	+∞
BFS	97.4	90.8	89.1	73.5	87.5	80.0	94.6	33.2	99.3	92.7	45.2	80.8	81.8	96.5	66.8	70.6	79.0	92.0
DFS	83.7	+∞	85.2	74.9	**70.6**	84.6	90.5	+∞	99.7	87.7	45.6	80.2	93.7	88.7	80.5	74.4	+∞	+∞
RANDOM	+∞	98.2	92.4	44.5	89.2	85.1	95.0	+∞	99.0	92.7	+∞	83.2	87.9	96.8	85.0	77.8	71.0	+∞
Saved req.	34.4	0.0	0.0	0.0	0.0	0.0	27.4	0.0	82.6	2.2	39.0	18.8	20.4	0.0	73.1	0.0	0.0	0.0
Lost targets	13.5	0.0	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	2.5	0.4	0.1	0.0	2.0	0.0	0.0	0.0

Table 3: Fraction of non-target volume retrieved by each crawler, before retrieving 90% of the total target volume

Crawler	*ab*	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*
SB-ORACLE	NA	NA	24.2	NA	56.3	24.6	49.2	NA	12.5	59.7	NA	22.9	29.5	48.0	33.2	30.2	NA	NA
SB-CLASS.	**20.4**	**21.4**	**29.5**	**29.1**	56.0	**29.0**	**49.2**	53.2	**23.6**	**64.7**	**18.6**	**23.1**	**34.5**	**49.5**	**34.9**	**33.2**	**32.7**	**35.8**
FOCUSED	+∞	+∞	85.2	97.0	76.3	74.7	86.4	+∞	67.3	73.8	66.8	72.2	84.9	72.7	49.8	80.3	+∞	+∞
TP-OFF	+∞	+∞	92.3	64.4	65.0	94.7	92.9	+∞	98.8	89.7	+∞	72.3	89.2	89.0	73.6	46.9	+∞	+∞
BFS	81.8	75.7	66.5	98.5	80.8	50.4	93.2	**3.6**	99.0	93.8	54.0	68.0	84.5	97.5	63.3	87.3	91.5	98.3
DFS	98.6	+∞	64.2	97.0	**45.0**	82.4	90.8	+∞	98.1	85.0	59.5	68.6	96.1	90.5	97.0	75.0	+∞	+∞
RANDOM	71.6	+∞	83.4	+∞	89.3	82.7	92.9	+∞	95.8	98.3	+∞	70.1	88.2	98.1	86.6	77.8	+∞	+∞

months to match our targets when all crawls were stopped (linear extrapolation); choosing FOCUSED would require 9 months. On *wh*, the best baseline (RANDOM) would take 3 months, and the worst (TP-OFF) 6 years. On *il*, the top-3 baselines fail to find additional targets in the last 650k iterations, while ours does until the end.

On other sites (*be*, *cn*, *cl*, *jp*, *qa*), while some baselines approach our crawler's performance, ours remains superior overall. Thus, **our approach outperforms all considered baselines on a wide variety of websites**, with respect to the size, depth, target distribution, etc. Small sites (*be*, *cl*, *cn*, *qa*) are traversed quickly, allowing less learning, yet our approach adapts effectively.

The offline-trained, tag-paths-based crawler TP-OFF [20] performs poorly, especially on *ed*, *il*, *jp*, *wh*, and *wo*. Learning from just 3 000 pages proves insufficient on large websites. On *cn*, TP-OFF underperforms BFS after the initial 3 000 pages, despite only three times more pages remaining. Increasing the training set could help but would approach the simplicity of a standard BFS crawler.

The focused-crawler FOCUSED is outperformed by ours on all websites, showing it is not adapted to our problem, given their different nature and different purpose. On 10 websites out of 18, it is even beaten by BFS, DFS or RANDOM.

Despite being given **three unfair advantages** (hand-picked keywords, access to training pages with target links, and an oracle for URL type prediction), **the topical crawler TRES fails on 9 out of 10 websites**, performing reasonably only on *oe* and only until it had to be stopped. More critically, **TRES cannot scale to medium or large sites**, even when run fully locally. Crawl iterations slow dramatically due to exhaustive feature-based evaluations during tree expansion.These results confirm that **topical crawlers are fundamentally ill-suited for direct target retrieval**, even when heavily adapted and unfairly favored.

Comparing our SB-CLASSIFIER with SB-ORACLE shows that **our classifier is close to the (virtual) perfect oracle**. Further analysis (our classifier's impact) is provided in the extended version [24].

4.6 Hyper-Parameters

We present the impact of three key hyper-parameters (α, n, and θ) on crawl performance. Table 4 reports the number of requests (resp. response volume for non-target pages) needed to retrieve 90% of the targets (resp. total target volume); full plots for various hyper-parameter values are in [24]. When varying one parameter, others remain at default values (Sec. 4.5). We also tested other parameters, such as the dimension D and those for projection to fixed-size vectors (Sec. 3.2), without a significant impact.

(1) *Impact of α.* α controls the exploration–exploitation trade-off (Sec. 3.2). Table 4 (top) shows results for $\alpha \in \{0.1, 2\sqrt{2}, 30\}$. Since optimality for $2\sqrt{2}$ is not guaranteed in our non-standard MAB environment, we also tested 0.1 and 30. Smaller α generally improves performance, with **best results for** $\alpha = 2\sqrt{2}$. Large α values, especially on websites with moderate or low rewards, cause excessive *exploration*, neglecting useful, already discovered actions.

(2) *Impact of n in n-grams for tag path vector representation.* For merging tag paths (Sec. 3.1), we tested $n \in \{1, 2, 3\}$ ($n = 1$ represents paths as sets of HTML tags; $n = 2$ is default). Table 4 (middle) shows that while $n = 1$ works well on some sites, 2 and 3 typically outperform it, confirming our hypothesis that groups of tag paths provide a better learning basis than sets of tags. We chose $n = 2$: although $n = 3$ performs similarly, the feature space grows exponentially with n, slowing training, so larger n were not tested.

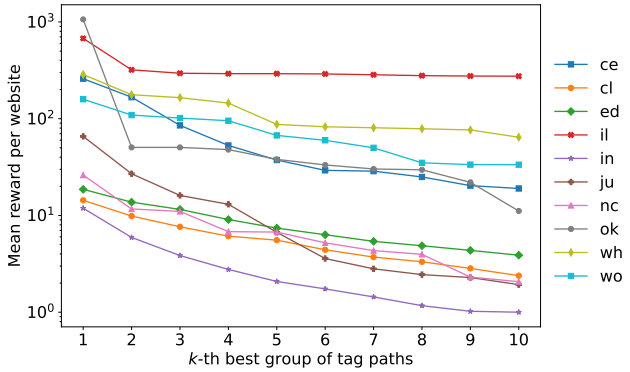


Figure 5: Mean rewards of the top-10 groups of tag paths for the ten selected websites from Figure 4 (log y-scale)

(3) *Impact of θ.* The similarity threshold (Sec. 3.1) affects how tag paths are grouped. We evaluated $\theta \in \{0.55, 0.75, 0.95\}$ (Table 4, bottom). High θ often performs poorly, especially when no similarity $> 95\%$ exists, as with websites adding unique IDs in tags. For example, on *ed* it caused an OOM error by creating as many actions as HTML pages. **Best results occur with $\theta \in \{0.55, 0.75\}$, with $\theta = 0.75$ usually superior**. On sites where $\theta = 0.55$ is better, $\theta = 0.75$ remains comparable, while the reverse is not true: e.g., on *ju*, $\theta = 0.75$ outperforms 0.55 by 40% in both requests and volume.

4.7 Effectiveness of SB Learning

We study the reward associated by our algorithm to tag paths. Figure 5 shows the mean reward of the top 10 path groups (logarithmic y-axis), for the ten websites in Figure 4.

We note that **top groups have high rewards**: across websites, the best group averages 258, followed by 89, 74, 67, and down to 41 for the 10th. This shows that **our Sleeping Bandit agent effectively identifies tag path groups leading to HTML pages with target links**. Although the first group often dominates, several groups still yield substantial reward, so restricting to the top one is insufficient. Comparing Figure 5 with the mean rewards for each website (on groups with non-zero rewards) in Table 5, the top 10 groups generally far exceed the overall mean (e.g., for *wo*, the the 10th group scores 33.5 v.s. 2.1 for the mean). This is also clear from the high STD values from Table 5 (over 20 times the mean for *wo*) implying **significant differences in the rewards of different groups**: rewards are not normally distributed across groups but more closely resemble a power law. Table 5 also shows large cross-website reward disparities, leading to the **impossibility of setting an optimal exploration–exploitation trade-off coefficient α.** This supports our pragmatic choice of $\alpha = 2\sqrt{2}$, effective in practice. These disparities further highlight strong heterogeneity in target concentration across websites, complementing Sec. 4.1.

Examining top-group tag paths often reveals clues about target-rich areas. In *nc*, a typical path includes `div.container.w-iap/div.iap-content`, linked to the *International Activities Program*, a program focused on collecting international education statistics. In *wo*, paths include `collections-sief`, related to the *Strategic Impact Evaluation Fund*, a trust fund producing data and reports. Some groups directly include target-retrieval keywords, e.g., `fr-link--download` in *ju* or `status-publish` in *ok*. Other websites' top paths are not human-interpretable yet still achieve high rewards (examples in [24]). This shows that our agent detects useful patterns even without semantic meaning, adapting to diverse structures without prior knowledge and supporting an online, per-website learning approach. This further demonstrates how our approach is language-independent: the RL agent learns structural patterns in HTML regardless of semantic meaning, enabling efficient target retrieval even on non-English or multilingual websites.

4.8 Early-Stopping

To prevent the crawler from continuing on websites with few or no remaining targets, we add an early-stopping mechanism. Every ν iterations, we compute a slope $\sigma = \frac{y_t - y_{t-\nu}}{\nu}$, representing the growth rate in the number y_t of targets at crawling step t. We then maintain an exponential moving average $\mu = \gamma \cdot \sigma + (1-\gamma) \cdot \mu$, with γ the decay rate. If μ stays below a threshold ϵ for κ consecutive slopes (i.e., $\kappa \cdot \nu$ iterations), crawling stops. Parameters ($\nu = 1000$, $\epsilon = 0.2, \gamma = 0.05, \kappa = 15$) provide a good balance between avoiding premature termination and efficiently stopping on exhausted sites.

The lower part of Table 2 (below the double rule) shows early-stopping results: the percentage of requests avoided and of targets missed. High values are better for the former, low for the latter. We observe three behaviors: (i) Medium to large websites where target discovery slows and stopping occurs effectively (*ab*, *ed*, *in*, *jp*, *ju*, *nc*, *ok*). Smaller sites take longer (in %) to stop due to the required $\kappa \cdot \nu$ iterations; (ii) Large websites with continuous target discovery where stopping conditions are never met before manual termination (*as*, *ce*, *il*, *oe*, *wh*, *wo*); (iii) Small sites (*be*, *cl*, *cn*, *qa*) where the crawl ends before early stopping can trigger within $\kappa \cdot \nu = 15$k iterations; this is not an issue, as these sites are quickly fully crawled.

5 Related Work

Web crawlers. We discuss several aspects of Web crawlers, see the extended version [24] for other relevant work, and for a summary table outlining the characteristics of existing focused crawlers.

(1) *Focused crawlers* prioritize some pages during the crawl, often based on predefined *topics*. Traditional focused crawlers [10, 19] mainly train a classifier to predict the likelihood that a hyperlink leads to a relevant page; we adapted this approach to our target retrieval in FOCUSED. Modern focused crawlers typically employ RL to adapt the crawl: we review them here. We distinguish two kinds of focused crawlers by their **focus**: most are *topical* crawlers [26, 28, 36, 55], which focus on a predefined *topic* via keywords or sample documents. *Non-topical* focused crawlers are rarer: besides this work (which focuses on *targets* of a given file type), examples include Anthelion [39], which focuses on semantic annotations in Web pages, and ACEBot [20], which accumulates as much diverse textual content as possible. Anthelion's focus is more specific than ours and ACEBot's focus differs but we adapted it for target retrieval in TP-OFF. Topical crawlers are ill-suited for target retrieval as shown in Sec. 4.5, since they target HTML content and require language-specific elements defining the topic. Our statistics dataset application requires crawling sites with content about different domains (economics, education, health, etc.) and in multiple languages (Sec. 4.1). Another key difference is that assumptions of

Table 4: Percentage of requests that an SB crawler with oracle performs to retrieve 90% of the targets (left of |). Right of |: percentage of the volume of non-target pages retrieved, before retrieving 90% of the total target volume. Hyper-parameter study on α (top), n (in n-grams, center), and θ (bottom), for the 11 fully-crawled websites.

Crawler	*be*	*cl*	*cn*	*ed*	*in*	*is*	*ju*	*nc*	*oe*	*ok*	*qa*
$\alpha = 0.1$	86.3 | 26.2	**75.9** | **42.3**	74.3 | 35.5	53.7 | 54.1	**9.8** | **10.2**	77.1 | 66.2	37.1 | 35.0	51.6 | **26.2**	**55.6** | **34.4**	14.3 | 33.2	**67.7** | 32.1
$\alpha = 2\sqrt{2}$	84.7 | **24.2**	76.4 | 56.3	**71.8** | **24.6**	**53.0** | 49.2	11.1 | 11.0	**74.2** | **58.9**	**35.0** | **22.9**	**51.4** | 29.5	59.2 | 48.0	**10.3** | **19.0**	68.9 | 33.9
$\alpha = 30$	**83.8** | 36.7	79.6 | 58.9	75.3 | 32.4	66.2 | **41.5**	11.6 | 11.8	80.9 | 66.4	43.3 | 28.8	67.3 | 29.5	68.8 | 72.9	36.7 | 71.3	71.8 | **30.4**
$n = 1$	84.5 | 27.1	77.2 | **48.5**	78.6 | 56.3	57.3 | 55.1	**9.9** | 10.7	78.2 | 69.6	35.7 | **17.6**	54.8 | 33.5	**52.6** | **28.1**	13.6 | 27.2	68.9 | 34.7
$n = 2$	84.7 | **24.2**	**76.4** | 56.3	71.8 | **24.6**	**53.0** | **49.2**	11.1 | 11.0	74.2 | 58.9	**35.0** | 22.9	51.4 | 29.5	59.2 | 48.0	10.3 | 19.0	**68.3** | **33.9**
$n = 3$	**84.1** | 32.8	78.2 | 51.2	**71.3** | 25.7	57.0 | 53.1	10.7 | **10.5**	**71.3** | **49.2**	37.0 | 26.9	**51.2** | **27.0**	79.6 | 79.0	**6.0** | **8.8**	70.0 | 34.9
$\theta = 0.55$	**81.2** | 42.0	76.8 | **50.5**	76.6 | 41.9	56.5 | 53.1	**8.2** | **9.4**	78.7 | 65.5	80.6 | 65.4	56.1 | 35.5	**52.4** | **30.9**	12.5 | 25.7	**67.8** | 26.0
$\theta = 0.75$	84.7 | **24.2**	**76.4** | 56.3	**71.8** | **24.6**	**53.0** | **49.2**	11.1 | 11.0	74.2 | 58.9	**35.0** | **22.9**	**51.4** | **29.5**	59.2 | 48.0	**10.3** | **18.7**	68.9 | 33.9
$\theta = 0.95$	82.4 | 47.7	84.3 | 72.1	73.1 | 44.7	OOM | OOM	9.8 | 11.0	**71.0** | **54.9**	73.3 | 66.5	57.3 | 33.2	90.2 | 87.2	12.4 | 19.0	68.3 | **25.9**

Table 5: Mean and STD of non-zero rewards of the learning agent on each website

	ab	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*
Mean	1.7	1.5	4.5	30.2	12.4	4.2	2.5	3.1	1.6	3.5	3.5	5.4	2.0	2.5	5.5	15.4	3.0	2.1
Std	16.8	5.35	20.9	290.3	2.8	8.9	7.1	53.9	4.2	11.1	17.4	10.5	8.7	9.3	13.9	18.8	22.0	43.5

topical locality [17] (similar-topic pages are close) and crawl direction [28] (page scores increase near topic-relevant pages) do not typically hold in our case. Existing systems differ in the way they use **learning**: many choose to have a simple single-state representation [26, 39, 55] in the form of classical MABs, while some introduce a multiple-state MDP representation as in [28] or [36] which directly extends on [28]. In order to support an MDP representation, [28, 36] have to define states that cannot just be the current state of the crawl but a summary thereof (as RL policy optimization requires visiting many times each state). For this, they use features characterizing the topic relevance of the last crawled page and its neighborhood. This does not have a direct equivalent in the setting of target retrieval (again, because of non-locality). We differ from existing work by choosing a SB representation, a mathematically well-founded way to both use a single state and model that not all actions are available at every step. It also allows faster convergence of action estimation than multiple-state representations. Finally, note that [20] does not use RL but an offline training on part of the website, as with TP-OFF. In most cases, and in this work, **actions** available to the focused crawler are retrieving the next URL. Two works differ: in [39], RL is only used for selecting *sites* to crawl the next URL from, URL selection following an independent function; in [55], in addition to URL retrieval, actions include using a SE. Finally, note that a major difference between this work and all other cited works except for [20] is that the task is to run a focused crawl of a given site – the more common approach is to crawl pages across sites, with no strict crawl **scope** restriction. In the application to the retrieval of statistics datasets, this is critical as we need to enable the user to only retrieve datasets from trustworthy sources.

(2) *Incremental* or *revisit policy-based* crawling views a website as an evolving collection of pages and revisits selectively to maximize updated content while minimizing unnecessary revisits. [11] presents core challenges, while [48] introduces an incremental version of Heritrix [41], used by the Internet Archive. More recent works propose learning-based revisit policies: [5, 16, 35] predict emergence of new outlinks using various feature types. Others use RL: [45] shows Thompson Sampling (TS) outperforms MAB alternatives and the Pham Crawler [43], while [34] introduces a learning algorithm for efficiently computing optimal revisit policies. While our method is a single-shot crawl for targets, we plan to build upon it to implement incremental strategies.

Tag paths. Tag paths, e.g., *root-to-link* paths in the DOM of each HTML page, have been exploited for Web crawling. [14, 15] take advantage of the website structure to cluster webpages, with applications to *Web scrapers* aiming to automatically extract and structure data from HTML pages. [20] directly uses paths in the DOM of each HTML page in order to do focused Web crawling.

6 Conclusion

We addressed the problem of scalable web data acquisition by developing an efficient crawling approach that aims to maximize the retrieval of targets (interesting pages or files) while minimizing computational resource usage in terms of time and bandwidth, and respecting crawling ethics. Our solution, based on RL, outperforms standard baselines on an heterogeneous set of websites.

In our future work, we would like to explore more complex reward functions than the simple number of targets reachable. Also, integrating deep-Web crawling techniques could enhance our crawler's ability to access data behind forms or within portals more efficiently, thereby further improving our web data acquisition at scale. Finally, while our crawler focuses on the initial acquisition of targets, it does not handle updates or newly published resources: an important limitation for statistical data journalism, where timely information is essential. We leave extending our crawler with *incremental revisits* for future work, combining the knowledge acquired by our RL-agent with existing re-crawling strategies.

Acknowledgments We thank Antoine Deiana (RadioFrance) for suggesting the SD retrieval problem while collaborating on [7].

Artifacts

The repository [24] provides a complete experiment reproducibility kit. The `README.md` file describes the repository contents and explains how it is organized. The kit mainly allows running the crawlers SB-ORACLE, SB-CLASSIFIER, FOCUSED, TP-OFF (sometimes referred to as `dom_off`), BFS, DFS, and RANDOM. Three execution modes are available:

- *Local crawling*: used when a website has already been fully replicated in a local database (see Sec. 4.4).
- *Online-to-local crawling*: creates a local copy of a website using a naive crawler.
- *Semi-online crawling*: first checks the local database and fetches the resource only if it is not present (applied to non-fully crawled websites, also described in Sec. 4.4).

The repository also contains code to reproduce the plots, information about the websites used in our experiments, and an extended version of this paper. The extended version includes: the proof of Prop. 4; additional details about the websites' characteristics; the MIME type list defining the targets used in the experiments; the initial keyword set for TRES; and a complete blocklist of URL extensions and MIME types. It also provides the plots of the 8 websites that could not be included in Figure 4; exhaustive plots from the hyper-parameter studies; extra examples of typical tag paths for 6 websites; additional results on URL classification quality; a visualization of the early-stopping mechanism for two websites; a table summarizing the main characteristics of related focused crawlers; and an extended discussion of other crawler types, as well as alternatives to AUER for MAB algorithms.

A second repository [25] contains a fork of the TRES [36] crawler code, adapted for the target retrieval task described in Sec. 4.3. This repository was originally published on GitHub under an anonymous profile for a previous double-blind submission; we explicitly confirm that we are its authors.

References

[1] Ayesh Alshukri, Frans Coenen, and Michele Zito. 2011. Incremental Web-Site Boundary Detection Using Random Walks. In *Machine Learning and Data Mining in Pattern Recognition - 7th International Conference, MLDM 2011, New York, NY, USA, August 30 - September 3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6871)*, Petra Perner (Ed.). Springer, 414–427. doi:10.1007/978-3-642-23199-5_31

[2] Fatma Arslan, Naeemul Hassan, Chengkai Li, and Mark Tremayne. 2020. A Benchmark Dataset of Check-Worthy Factual Claims. In *Proceedings of the Fourteenth International AAAI Conference on Web and Social Media, ICWSM 2020, Held Virtually, Original Venue: Atlanta, Georgia, USA, June 8-11, 2020*, Munmun De Choudhury, Rumi Chunara, Aron Culotta, and Brooke Foucault Welles (Eds.). AAAI Press, 821–829. https://ojs.aaai.org/index.php/ICWSM/article/view/7346

[3] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. 2002. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning* 47, 2 (2002), 235–256.

[4] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. DBpedia: A Nucleus for a Web of Open Data. In *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007 (Lecture Notes in Computer Science, Vol. 4825)*, Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux (Eds.). Springer, 722–735. doi:10.1007/978-3-540-76298-0_52

[5] Konstantin Avrachenkov, Kishor Patil, and Gugan Thoppe. 2022. Online algorithms for estimating change rates of web pages. *Perform. Evaluation* 153 (2022), 102261. doi:10.1016/J.PEVA.2021.102261

[6] Oana Balalau, Simon Ebel, Helena Galhardas, Théo Galizzi, and Ioana Manolescu. 2024. STaR: Space and Time-aware Statistic Query Answering. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, CIKM 2024, Boise, ID, USA, October 21-25, 2024*, Edoardo Serra and Francesca Spezzano (Eds.). ACM, 5190–5194. doi:10.1145/3627673.3679209

[7] Oana Balalau, Simon Ebel, Théo Galizzi, Ioana Manolescu, Quentin Massonnat, Antoine Deiana, Emilie Gautreau, Antoine Krempf, Thomas Pontillon, Gérald Roux, and Joanna Yakin. 2022. Statistical Claim Checking: StatCheck in Action. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, Mohammad Al Hasan and Li Xiong (Eds.). ACM, 4798–4802. doi:10.1145/3511808.3557198

[8] Léon Bottou. 2010. Large-Scale Machine Learning with Stochastic Gradient Descent. In *19th International Conference on Computational Statistics, COMPSTAT 2010, Paris, France, August 22-27, 2010 - Keynote, Invited and Contributed Papers*, Yves Lechevallier and Gilbert Saporta (Eds.). Physica-Verlag, 177–186. doi:10.1007/978-3-7908-2604-3_16

[9] Tien Duc Cao, Ioana Manolescu, and Xavier Tannier. 2018. Searching for Truth in a Database of Statistics. In *Proceedings of the 21st International Workshop on the Web and Databases, Houston, TX, USA, June 10, 2018*. ACM, 4:1–4:6. doi:10.1145/3201463.3201467

[10] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. 1999. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Comput. Networks* 31, 11-16 (1999), 1623–1640. doi:10.1016/S1389-1286(99)00052-3

[11] Junghoo Cho and Hector Garcia-Molina. 2000. The Evolution of the Web and Implications for an Incremental Crawler. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter, and Kyu-Young Whang (Eds.). Morgan Kaufmann, 200–209. http://www.vldb.org/conf/2000/P200.pdf

[12] Martin Pekár Christensen, Aristotelis Leventidis, Matteo Lissandrini, Laura Di Rocco, Renée J. Miller, and Katja Hose. 2025. Fantastic Tables and Where to Find Them: Table Search in Semantic Data Lakes. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.). OpenProceedings.org, 397–410. doi:10.48786/EDBT.2025.32

[13] Philipp Christmann, Rishiraj Saha Roy, and Gerhard Weikum. 2024. CompMix: A Benchmark for Heterogeneous Question Answering. In *Companion Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Roy Ka-Wei Lee, Ravi Kumar, and Hady W. Lauw (Eds.). ACM, 1091–1094. doi:10.1145/3589335.3651444

[14] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. 2003. Fine-grain web site structure discovery. In *Fifth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2003), New Orleans, Louisiana, USA, November 7-8, 2003*, Roger H. L. Chiang, Alberto H. F. Laender, and Ee-Peng Lim (Eds.). ACM, 15–22. doi:10.1145/956699.956703

[15] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. 2005. Clustering Web pages based on their structure. *Data Knowl. Eng.* 54, 3 (2005), 279–299. doi:10.1016/J.DATAK.2004.11.004

[16] Thi Kim Nhung Dang, Doina Bucur, Berk Atil, Guillaume Pitel, Frank Ruis, Hamid Reza Kadkhodaei, and Nelly Litvak. 2023. Look back, look around: A systematic analysis of effective predictors for new outlinks in focused Web crawling. *Knowl. Based Syst.* 260 (2023), 110126. doi:10.1016/J.KNOSYS.2022.110126

[17] Brian D. Davison. 2000. Topical locality in the Web. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, Emmanuel J. Yannakoudakis, Nicholas J. Belkin, Peter Ingwersen, and Mun-Kew Leong (Eds.). ACM, 272–279. doi:10.1145/345508.345597

[18] Yuhao Deng, Chengliang Chai, Lei Cao, Qin Yuan, Siyuan Chen, Yanrui Yu, Zhaoze Sun, Junyi Wang, Jiajun Li, Ziqi Cao, Kaisen Jin, Chi Zhang, Yuqing Jiang, Yuanfang Zhang, Yuping Wang, Ye Yuan, Guoren Wang, and Nan Tang. 2024. LakeBench: A Benchmark for Discovering Joinable and Unionable Tables in Data Lakes. *Proc. VLDB Endow.* 17, 8 (2024), 1925–1938. doi:10.14778/3659437.3659448

[19] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C Lee Giles, Marco Gori, et al. 2000. Focused Crawling Using Context Graphs. In *VLDB*. 527–534.

[20] Muhammad Faheem and Pierre Senellart. 2015. Adaptive Web Crawling Through Structure-Based Link Classification. In *Digital Libraries: Providing Quality Information - 17th International Conference on Asia-Pacific Digital Libraries, ICADL 2015, Seoul, Korea, December 9-12, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9469)*, Robert B. Allen, Jane Hunter, and Marcia Lei Zeng (Eds.). Springer, 39–51. doi:10.1007/978-3-319-27974-9_5

[21] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-aware Dataset Discovery from Data Lakes with Contextualized Column-based Representation Learning. *Proc. VLDB Endow.* 16, 7 (2023), 1726–1739. doi:10.14778/3587136.3587146

[22] Tim Furche, Giovanni Grasso, Andrey Kravchenko, and Christian Schallhart. 2012. Turn the Page: Automated Traversal of Paginated Websites. In *Web Engineering - 12th International Conference, ICWE 2012, Berlin, Germany, July 23-27, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7387)*, Marco Brambilla, Takehiro Tokuda, and Robert Tolksdorf (Eds.). Springer, 332–346. doi:10.1007/978-3-642-31753-8_27

[23] M. R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[24] Antoine Gauquier, Ioana Manolescu, and Pierre Senellart. 2025. Efficient Crawling for Scalable Web Data Acquisition (experiment reproducibility kit). https://github.com/AntoineGauquier/efficient_crawler_for_scalable_web_data_acquisition/.

[25] Antoine Gauquier, Ioana Manolescu, and Pierre Senellart. 2025. TRES System Adapted to the Target Retrieval Task. https://github.com/anonymous-authors-301319/tres_modified_for_target_retrieval.

[26] Georges Gouriten, Silviu Maniu, and Pierre Senellart. 2014. Scalable, generic, and adaptive systems for focused crawling. In *25th ACM Conference on Hypertext and Social Media, HT '14, Santiago, Chile, September 1-4, 2014*, Leo Ferres, Gustavo Rossi, Virgílio A. F. Almeida, and Eelco Herder (Eds.). ACM, 35–45. doi:10.1145/2631775.2631795

[27] Yue Guan, Anuradha M. Annaswamy, and H. Eric Tseng. 2020. Towards Dynamic Pricing for Shared Mobility on Demand using Markov Decision Processes and Dynamic Programming. In *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 1–7. doi:10.1109/ITSC45102.2020.9294685

[28] Miyoung Han, Pierre-Henri Wuillemin, and Pierre Senellart. 2018. Focused Crawling Through Reinforcement Learning. In *Web Engineering - 18th International Conference, ICWE 2018, Cáceres, Spain, June 5-8, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10845)*, Tommi Mikkonen, Ralf Klamma, and Juan Hernández (Eds.). Springer, 261–278. doi:10.1007/978-3-319-91662-0_20

[29] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. 2021. Open Domain Question Answering over Tables via Dense Retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 512–519. doi:10.18653/V1/2021.NAACL-MAIN.43

[30] Madelon Hulsebos, Wenjing Lin, Shreya Shankar, and Aditya G. Parameswaran. 2024. It Took Longer than I was Expecting: Why is Dataset Search Still so Hard?. In *Proceedings of the 2024 Workshop on Human-In-the-Loop Data Analytics, HILDA 24, Santiago, Chile, 14 June 2024*, Jean-Daniel Fekete, Behrooz Omidvar-Tehrani, Kexin Rong, and Roee Shraga (Eds.). ACM, 1–4. doi:10.1145/3665939.3665959

[31] Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. 2013. *An introduction to statistical learning*. Vol. 112. Chapter 4.3.

[32] Georgios Karagiannis, Mohammed Saeed, Paolo Papotti, and Immanuel Trummer. 2020. Scrutinizer: A Mixed-Initiative Approach to Large-Scale, Data-Driven Claim Verification. *Proc. VLDB Endow.* 13, 11 (2020), 2508–2521. http://www.vldb.org/pvldb/vol13/p2508-karagiannis.pdf

[33] Robert D. Kleinberg, Alexandru Niculescu-Mizil, and Yogeshwer Sharma. 2008. Regret Bounds for Sleeping Experts and Bandits. In *21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, July 9-12, 2008*, Rocco A. Servedio and Tong Zhang (Eds.). Omnipress, 425–436. http://colt2008.cs.helsinki.fi/papers/114-Kleinberg.pdf

[34] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric Horvitz. 2019. Staying up to Date with Online Content Changes Using Reinforcement Learning for Scheduling. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 579–589. https://proceedings.neurips.cc/paper/2019/hash/ad13a2a07ca4b7642959dc0c4c740ab6-Abstract.html

[35] Andrey Kolobov, Yuval Peres, Eyal Lubetzky, and Eric Horvitz. 2019. Optimal Freshness Crawl Under Politeness Constraints. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, and Falk Scholer (Eds.). ACM, 495–504. doi:10.1145/3331184.3331241

[36] Andreas Kontogiannis, Dimitrios Kelesis, Vasilis Pollatos, Georgios Paliouras, and George Giannakopoulos. 2021. Tree-based Focused Web Crawling with Reinforcement Learning. *CoRR* abs/2112.07620 (2021). arXiv:2112.07620 https://arxiv.org/abs/2112.07620

[37] Colin Lockard, Prashant Shiralkar, Xin Luna Dong, and Hannaneh Hajishirzi. 2020. Web-scale Knowledge Collection. In *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, James Caverlee, Xia (Ben) Hu, Mounia Lalmas, and Wei Wang (Eds.). ACM, 888–889. doi:10.1145/3336191.3371878

[38] Yury A. Malkov and Dmitry A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR* abs/1603.09320 (2016). arXiv:1603.09320 http://arxiv.org/abs/1603.09320

[39] Robert Meusel, Peter Mika, and Roi Blanco. 2014. Focused Crawling for Structured Data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, Jianzhong Li, Xiaoyang Sean Wang, Minos N. Garofalakis, Ian Soboroff, Torsten Suel, and Min Wang (Eds.). ACM, 1039–1048. doi:10.1145/2661829.2661902

[40] Gengxin Miao, Jun'ichi Tatemura, Wang-Pin Hsiung, Arsany Sawires, and Louise E. Moser. 2009. Extracting data records from the web using tag path clustering. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl (Eds.). ACM, 981–990. doi:10.1145/1526709.1526841

[41] Gordon Mohr, Michael Stack, Igor Rnitovic, Dan Avery, and Michele Kimpton. 2004. Introduction to Heritrix. In *4th International Web Archiving Workshop*. 109–115.

[42] Taylor Orth. 2022. From millionaires to Muslims, small subgroups of the population seem much larger to many Americans. https://today.yougov.com/politics/articles/41556-americans-misestimate-small-subgroups-population.

[43] Kien Pham, Aécio S. R. Santos, and Juliana Freire. 2018. Learning to Discover Domain-Specific Web Content. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, Yi Chang, Chengxiang Zhai, Yan Liu, and Yoelle Maarek (Eds.). ACM, 432–440. doi:10.1145/3159652.3159724

[44] Mohammed Saeed and Paolo Papotti. 2021. Fact-Checking Statistical Claims with Tables. *IEEE Data Eng. Bull.* 44, 3 (2021), 27–38. http://sites.computer.org/debull/A21sept/p27.pdf

[45] Peter Schulam and Ion Muslea. 2023. Improving the Exploration/Exploitation Trade-Off in Web Content Discovery. In *Companion Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben (Eds.). ACM, 1183–1189. doi:10.1145/3543873.3587574

[46] SDMX Community. 2024. SDMX Technical Specifications. https://sdmx.org/?page_id=5008.

[47] Pierre Senellart. 2005. Identifying Websites with Flow Simulation. In *Web Engineering, 5th International Conference, ICWE 2005, Sydney, Australia, July 27-29, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3579)*, David B. Lowe and Martin Gaedke (Eds.). Springer, 124–129. doi:10.1007/11531371_18

[48] Kristinn Sigurðsson. 2005. Incremental crawling with Heritrix. In *Proceedings of the 5th International Web Archiving Workshop*. Vienna, Austria. http://iwaw.europarchive.org/05/papers/iwaw05-sigurdsson.pdf

[49] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement learning - an introduction*. MIT Press. http://www.incompleteideas.net/book/first/the-book.html

[50] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. 2020. YAGO 4: A Reason-able Knowledge Base. In *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings (Lecture Notes in Computer Science, Vol. 12123)*, Andreas Harth, Sabrina Kirrane, Axel-Cyrille Ngonga Ngomo, Heiko Paulheim, Anisa Rula, Anna Lisa Gentile, Peter Haase, and Michael Cochez (Eds.). Springer, 583–596. doi:10.1007/978-3-030-49461-2_34

[51] Andreas Vlachos and Sebastian Riedel. 2015. Identification and Verification of Simple Claims about Statistical Properties. In *EMNLP*.

[52] Qiming Wang and Raul Castro Fernandez. 2023. Solo: Data Discovery Using Natural Language Questions Via A Self-Supervised Approach. *Proc. ACM Manag. Data* 1, 4 (2023), 262:1–262:27. doi:10.1145/3626756

[53] Gerhard Weikum, Xin Luna Dong, Simon Razniewski, and Fabian M. Suchanek. 2021. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. *Found. Trends Databases* 10, 2-4 (2021), 108–490. doi:10.1561/1900000064

[54] WHATWG. 2024. DOM: Living Standard. https://dom.spec.whatwg.org/.

[55] Haoxiang Zhang, Aécio S. R. Santos, and Juliana Freire. 2021. DSDD: Domain-Specific Dataset Discovery on the Web. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 2527–2536. doi:10.1145/3459637.3482427

A Supplementary material for Section 2 (Problem Statement and Modeling)

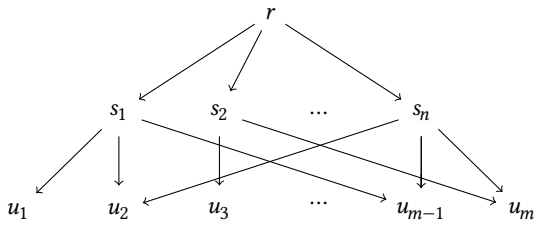


Figure 6: Graphical summarization of the graph $G_{\mathbf{sc}}$

A.1 Graph Crawling Problem

Proposition 4. *Given a website graph $G = (V, E, r, \omega, \lambda)$, a subset $V^* \subseteq V$ and some $B \in \mathbb{R}^+$, determining whether there exists a crawl $T = (V', E')$ of G such that $V^* \subseteq V'$ and $\omega(T) \leq B$ is NP-complete; hardness holds even when ω is a constant function. This holds even when nodes in V^* do not have any outgoing links in G.*

Proof. To show NP-completeness, we must show that the problem belongs to NP and is NP-hard.

Let us start with the upper bound. Given a graph $G = (V, E, r, \omega, \lambda)$, we guess a subgraph $T = (V', E')$ of G (which is a polynomial-sized guess). In polynomial time, we check whether T is a r-rooted tree (i.e., whether it is connected, includes r, r has indegree 0 and other nodes indegree 1), we check that V' contains all nodes of V^*, and we check that $\omega(T) \leq B$. We accept if and only if these conditions are all satisfied. This yields a nondeterministic polynomial-time algorithm, meaning the problem is in NP.

We now move to the lower bound. Our crawling problem can be seen as a directed variant of the well-known NP-complete *Steiner Tree* [GJ79] problem. NP-hardness of the directed Steiner tree problem is mentioned in the literature (see, e.g., [WW16]), but as it is not formally shown there, we prefer for completeness of the presentation reducing from the set cover problem, a classic NP-hard problem [GJ79]. We denote $\mathcal{U} = \{u_1, \ldots, u_m\}$ a set of m elements called the universe. We also define a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of n non-empty subsets, each of them containing some elements of $\mathcal{U}$, such that:

$$\bigcup_{s \in \mathcal{S}} s = \mathcal{U}.$$

In its decision version, the set cover consists in given such a universe and collection, given a natural integer B, determining whether there exists a *cover* $\mathcal{C} \subseteq \{s_1, \ldots, s_n\}$ such that $|\mathcal{C}| \leq B$ and:

$$\bigcup_{s \in \mathcal{C}} s = \mathcal{U}.$$

We now propose a polynomial-time many-one reduction of the set cover problem to an instance of the graph crawling problem. We create a website graph $G_{\text{sc}} = (V_{\text{sc}}, E_{\text{sc}}, r, \omega, \lambda)$ as follows. We set V_{sc} to be $\{u_1, \ldots, u_m, r, s_1, \ldots, s_n\}$, including representations for every element of the universe $\mathcal{U}$, every set of the collection $\mathcal{S}$, as well as a distinct root r (by abuse of notation, we do not distinguish between elements of $\mathcal{U}$, $\mathcal{S}$ and the way they are represented in V_{sc}). We define E_{sc} as $\{(r, s_i) \mid i \in \{1, \ldots, n\}\} \cup \{(s_i, u) \mid u \in s_i, i \in \{1, \ldots, n\}\}$. In other words, in G_{sc} from the origin (root) r, we model each element of $\mathcal{S}$ as a vertex, that can be reached following a dedicated (directed) edge. Finally, for each new vertex $s_i \in \mathcal{S}$, we have as many outgoing edges as there are elements of $\mathcal{U}$ in s_i. Finally, we set ω to be the constant function that assigns cost 1 to every vertex, and λ to be some constant function. The result is a graph in the form of a tree of depth 2, depicted in Figure 6. We fix V^* to be $\mathcal{U}$. Observe that nodes in $\mathcal{U}$ do not have any outgoing links. We state that there exists $\mathcal{C} \subseteq \{s_1, \ldots, s_n\}$ such that $|\mathcal{C}| \leq B$ and $\bigcup_{i \in \mathcal{C}} s_i = \mathcal{U}$ if and only if there exists a crawl T_{sc} of G_{sc} containing all elements of V^* and of total cost $\omega(T_{\text{sc}}) \leq |\mathcal{U}| + B + 1$.

Let us explain why this reduction is polynomial-time. In set cover, the universe $\mathcal{U}$ can be described by the number m of its elements, with representation size $\Theta(\log m)$. Each set of $\mathcal{S}$ needs to list every element within this set, so a set s_i has representation size $\Theta(\log m \times |s_i|)$. Finally, B has representation size $\Theta(\log B)$. This yields a total input size of $\Theta((\log m)\left(\sum_{i=1}^{n} |s_i| + 1\right) + \log B)$. Note that $\sum_{i=1}^{n} |s_i| \geq \max(m, n)$ so this is $\Omega(\max(m, n) \log m + \log B)$. But then, the construction depicted in Figure 6 can clearly be done in time polynomial in m and n (namely, in $O(m \times n)$ in the worst case where every set of the collection contains every element). The reduction is therefore polynomial-time.

We now proceed to show equivalence between the initial problem known to be NP-hard (set cover) and the graph crawling instance presented above. First, suppose that there exists $\mathcal{C} \subseteq \{s_1, \ldots, s_n\}$ such that $|\mathcal{C}| \leq B$ and $\bigcup_{i \in \mathcal{C}} s_i = \mathcal{U}$. Then consider the crawl T_{sc} of G_{sc} formed by including r, every element of $\mathcal{C}$ using the edge from r to that element, and every edge from an element of $\mathcal{C}$ to an element of $\mathcal{U}$. Since $\mathcal{C}$ is a cover, this includes all elements of $\mathcal{U}$. The total cost of this crawl $\omega(T_{\text{sc}}) = 1 + |\mathcal{C}| + |\mathcal{U}| \leq |\mathcal{U}| + B + 1$.

Now suppose that there exists a crawl T_{sc} of G_{sc} of total cost $\omega(T_{\text{sc}}) \leq |\mathcal{U}| + B + 1$. Note that by definition of ω, the cost is just the number of nodes in T_{sc}, and this crawl necessarily includes the root r as well as all vertices of $\mathcal{U}$. The remaining $\leq B$ vertices are therefore vertices of $\mathcal{S}$. We pose $\mathcal{C}$ to be those. Then $|\mathcal{C}| \leq B$ and since T_{sc} is a crawl, for every $u \in \mathcal{U}$, there exists at least one $s \in \mathcal{C}$ such that the edge (s, u) is in T_{sc}, meaning that $u \in s$. We indeed have $\mathcal{U} = \bigcup_{s \in \mathcal{C}} s$. □

A.2 Data Acquisition as Graph Crawling

Here is the full list of the 38 MIME types used to identify targets in our implementation:

```
application/csv
application/json
application/msword
application/octet-stream
application/pdf
application/rdf+xml
application/rss+xml
application/vnd.ms-excel
application/vnd.ms-excel.sheet.macroenabled.12
application/vnd.oasis.opendocument.presentation
application/vnd.oasis.opendocument.spreadsheet
application/vnd.oasis.opendocument.text
application/vnd.openxmlformats-officedocument.presentationml.presentation
application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
application/vnd.openxmlformats-officedocument.wordprocessingml.document
application/vnd.openxmlformats-officedocument.wordprocessingml.template
application/vnd.rar
application/x-7z-compressed
application/x-csv
application/x-gtar
application/x-gzip
application/xml
application/x-pdf
application/x-rar-compressed
application/x-tar
application/x-yaml
application/x-zip-compressed
application/yaml
application/zip
application/zip-compressed
text/comma-separated-values
text/csv
text/json
text/plain
text/x-comma-separated-values
text/x-csv
text/x-yaml
text/yaml
```

B Supplementary material for Section 4 (Experimental Results)

B.1 Websites

To further emphasize the diversity of crawling efforts, we manually analyzed the types of pages where targets are typically found. In *ju*, targets appear under various resource lists and data portals disseminated across the website. Some are huge (e.g., https://www.justice.gouv.fr/documentation/bulletin-officiel), while others are smaller (e.g., https://www.justice.gouv.fr/documentation/ressources/conventions-judiciaires-dinteret-public). Some require multi-step navigation to reach deeper pages that contain one or a few target links. In *il*, most targets are found in small, dispersed data catalogs: some gathered in "ILOSTAT" (https://ilostat.ilo.org/), others accessible only through further exploration. These targets are generally downloadable directly from the catalogs. In *wh*, the majority of targets require multi-step navigation through a hierarchy of subjects and sub-subjects, with only a subset leading to targets: efficient crawling thus requires quickly identifying interesting parts. In *nc*, all previously mentioned typologies are present: data portals (e.g., https://nces.ed.gov/national-center-education-statistics-nces/products), exhaustive lists (e.g., https://nces.ed.gov/surveys/spp/results.asp), and deep pages requiring multi-step navigation.

B.2 Baselines

We use the following list of keywords as initial input to TRES to train its classifier:

pdf	document	statistics	download CSV
xls	report	article	download PDF
csv	publication	paper	download XLS
tar	dataset	metadata	dataset download
zip	data	fact	attached document
rar	download	download file	official documents
rdf	archive	download document	browse files
json	spreadsheet	available for download	download statistics
doc	table	access data	download article
xml	list	view report	annual report
yaml	resource	get dataset	white paper
txt	annex	data file	technical documentation
tsv	supplement	read more	technical report
ppt	attachment	resource list	raw data
ods	proceedings	get document	metadata file
dta	survey	download pulication	open data
7z	material	document archive	fact sheet
ttl	output	supporting materials	
file	content	export data	

B.3 Comparison with Baselines

In the experiments, we apply filters on MIME types and URL extensions to avoid downloading multimedia content. The MIME types in the blocklist are those for multimedia content, namely `image/*`, `audio/*`, and `video/*`. The associated URL extensions are the following:

.3g2	.asx	.dtshd	.flac	.ief	.lvp	.mj2	.mpg4	.pef	.ras	.tiff	.xbm
.3ga	.avi	.dwg	.fli	.jfi	.m1v	.mjp2	.mpg	.pgm	.raw	.ts	.xif
.3gp2	.avif	.dxf	.flv	.jfif-	.m2a	.mka	.mpga	.pic	.rgb	.viv	.xpm
.3gp	.avifs	.ecelp4800	.fpx	tbnl	.m2v	.mkv	.mrw	.pjpg	.rlc	.wav	.xwd
.3gpa	.bmp	.ecelp7470	.fst	.jfif	.m3a	.mmr	.mxu	.png	.rmi	.wax	
.3gpp2	.btif	.ecelp9600	.fvt	.jif	.m3u	.mov	.nef	.pnm	.rmp	.wbmp	
.3gpp	.cgm	.eol	.g3	.jpe	.m4a	.movie	.npx	.ppm	.rw2	.weba	
.aac	.cmx	.erf	.gif	.jpeg	.m4b	.mp2	.oga	.psd	.rwl	.webm	
.aacp	.cr2	.f4v	.h261	.jpg	.m4p	.mp2a	.ogg	.ptx	.snd	.webp	
.adp	.crw	.fbs	.h263	.jpgm	.m4r	.mp3	.ogv	.pya	.spx	.wm	
.aff	.dcr	.fh4	.h264	.jpgv	.m4u	.mp4	.opus	.pyv	.sr2	.wma	
.aif	.djv	.fh5	.heic	.jpm	.m4v	.mp4v	.orf	.qt	.srf	.wmv	
.aiff	.djvu	.fh7	.heif	.k25	.mdi	.mpa	.pbm	.ra	.svg	.wmx	
.arw	.dng	.fh	.icns	.kar	.mid	.mpe	.pct	.raf	.svgz	.wvx	
.asf	.dts	.fhc	.ico	.kdc	.midi	.mpeg	.pcx	.ram	.tif	.x3f	

Note that URL extension filters are not useful on all websites (on some, URLs don't usually include any extension, see the discussion at the beginning of Section 3.3).

Figure 7 completes Figure 4 with the eight remaining websites plots that could not fit in the paper (see Table 1 for characteristics of websites). Figures 8 to 13 present exhaustive graphical results of the hyper-parameters studies conducted on the 11 fully-crawled websites.

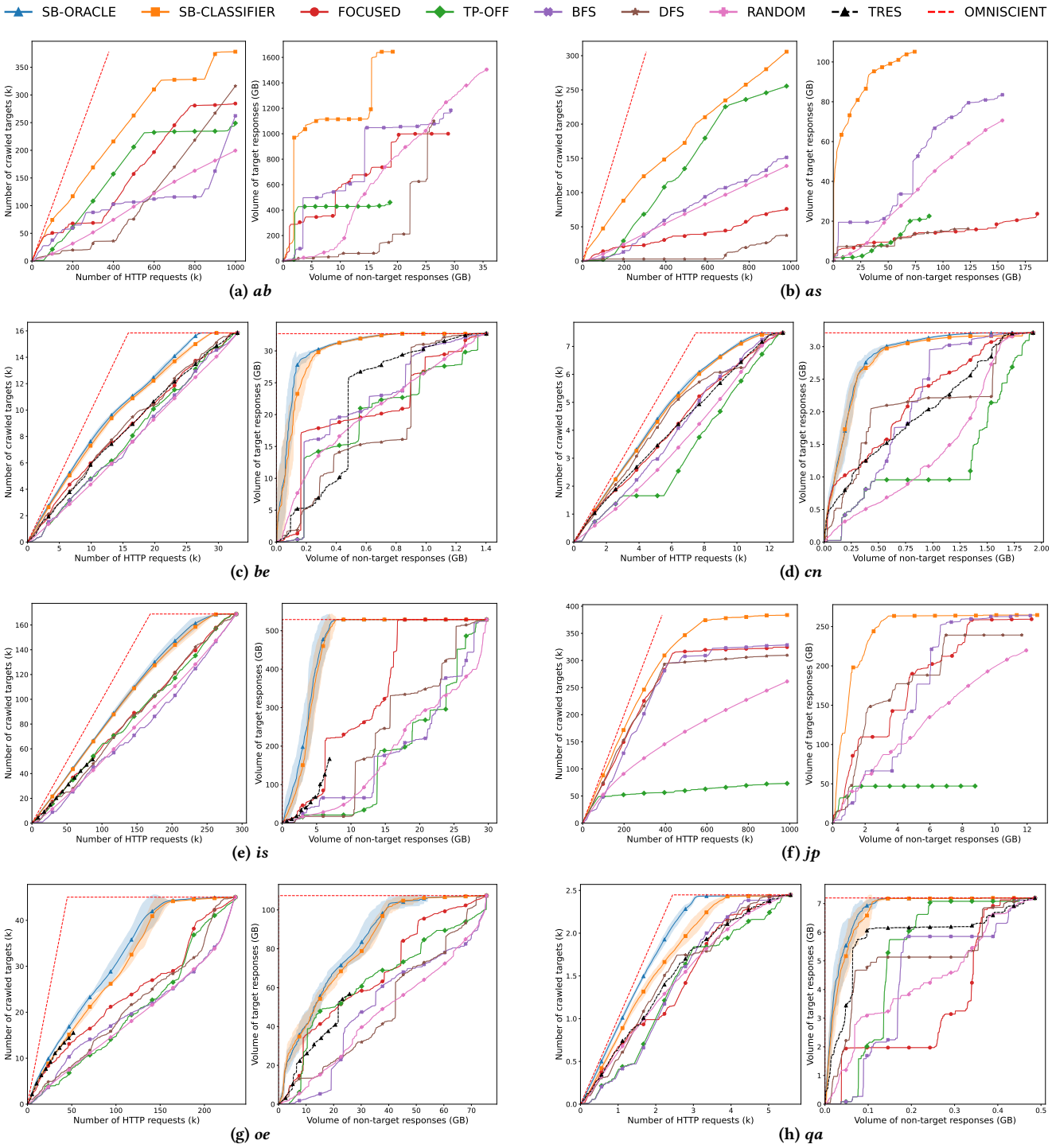


Figure 7: Comparison of different crawler performance for the 8 websites not presented in Figure 4 due to space reasons; for TRES, experiments are only shown for fully-crawled websites

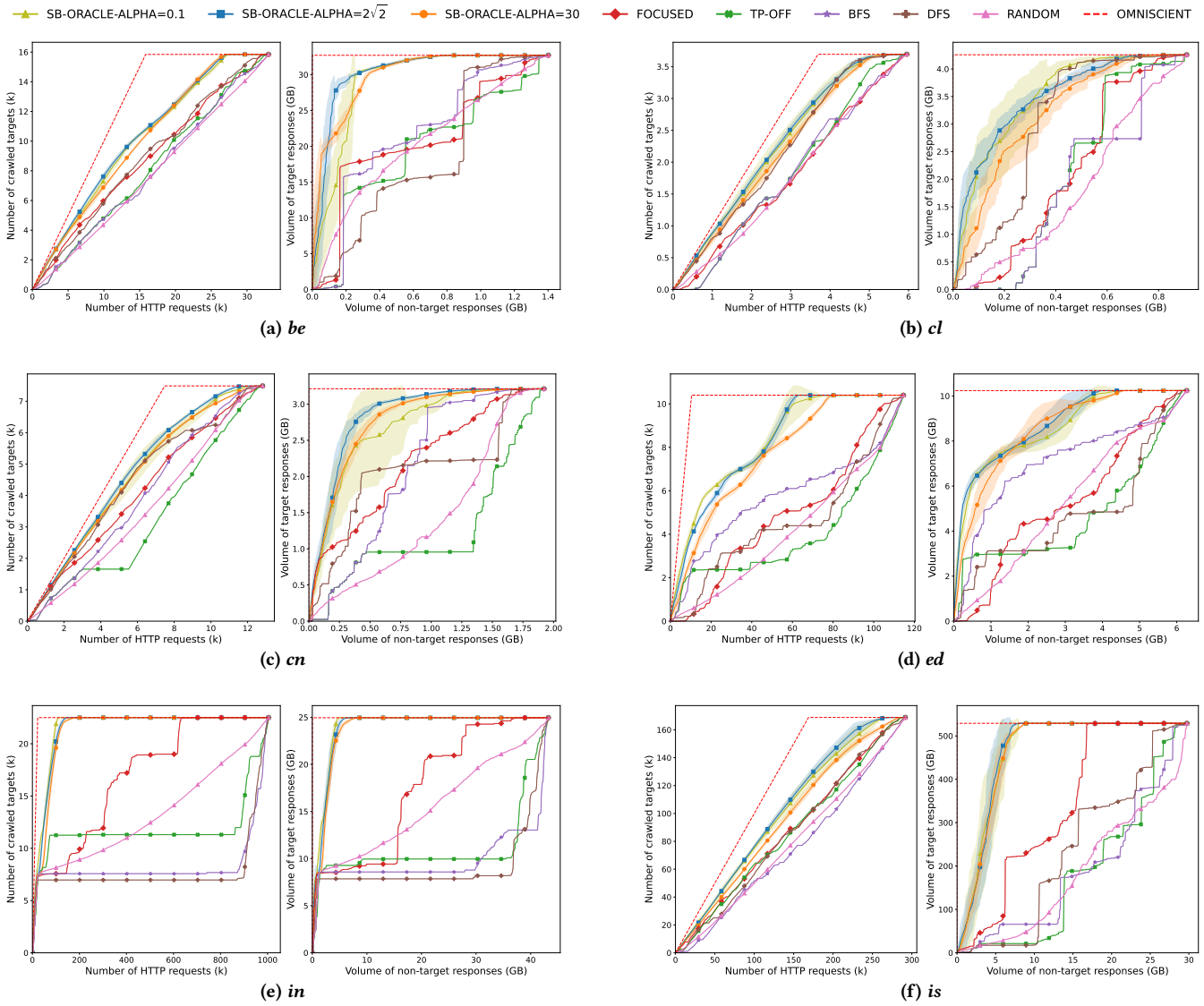
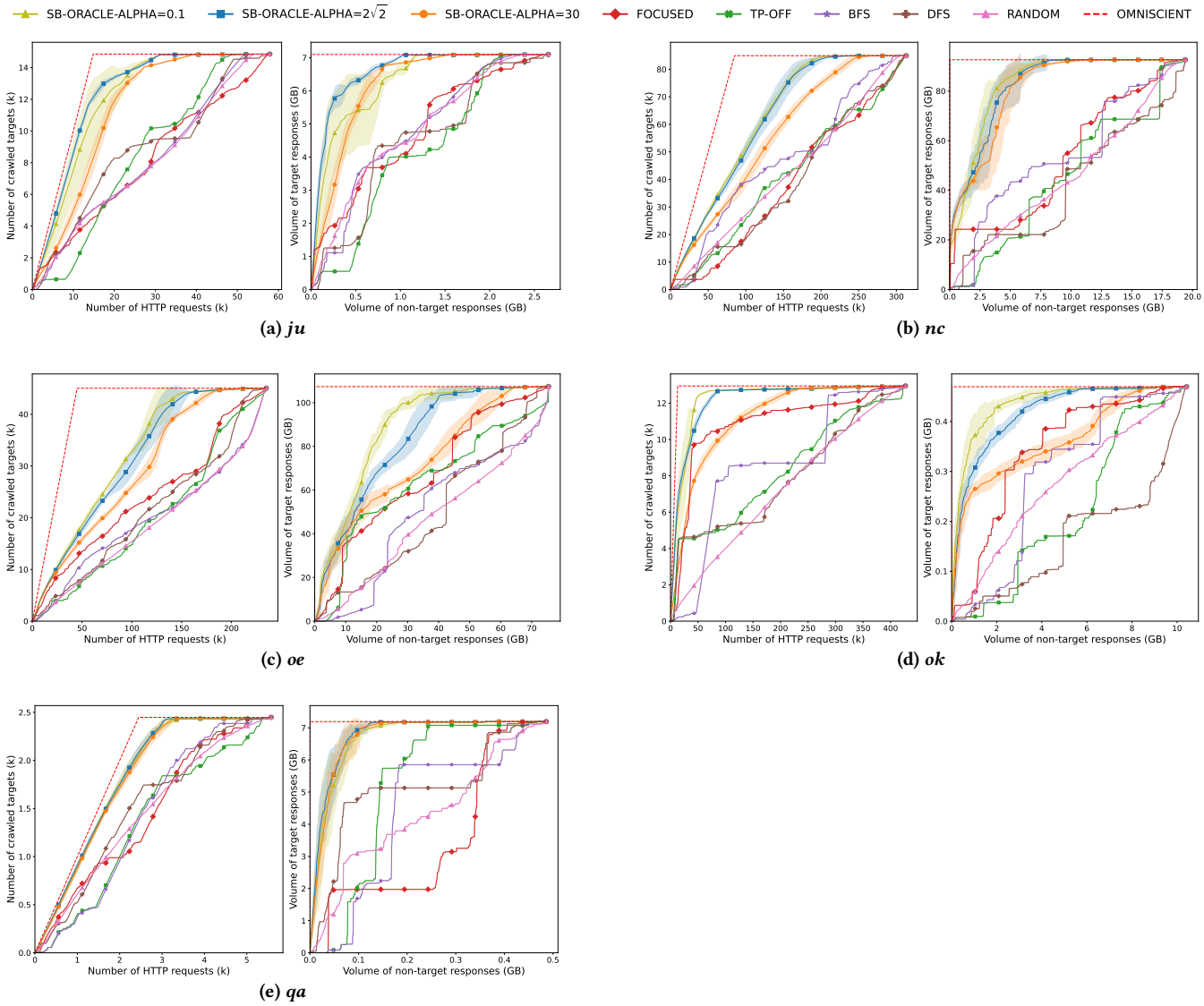


Figure 8: Crawler performance for hyper-parameter study on exploration–exploitation coefficient α, for websites *be*, *cl*, *cn*, *ed*, *in*, and *is*

Especially, Figures 8 and 9 study the exploration–exploitation coefficient α, Figures 10 and 11 study the choice of n in n-grams used in tag paths vector representation, and Figures 12 and 13 study the similarity threshold θ.



(a) *ju* (b) *nc* (c) *oe* (d) *ok* (e) *qa*

Figure 9: Crawler performance for hyper-parameter study on exploration–exploitation coefficient α, for websites *ju*, *nc*, *oe*, *ok*, and *qa*

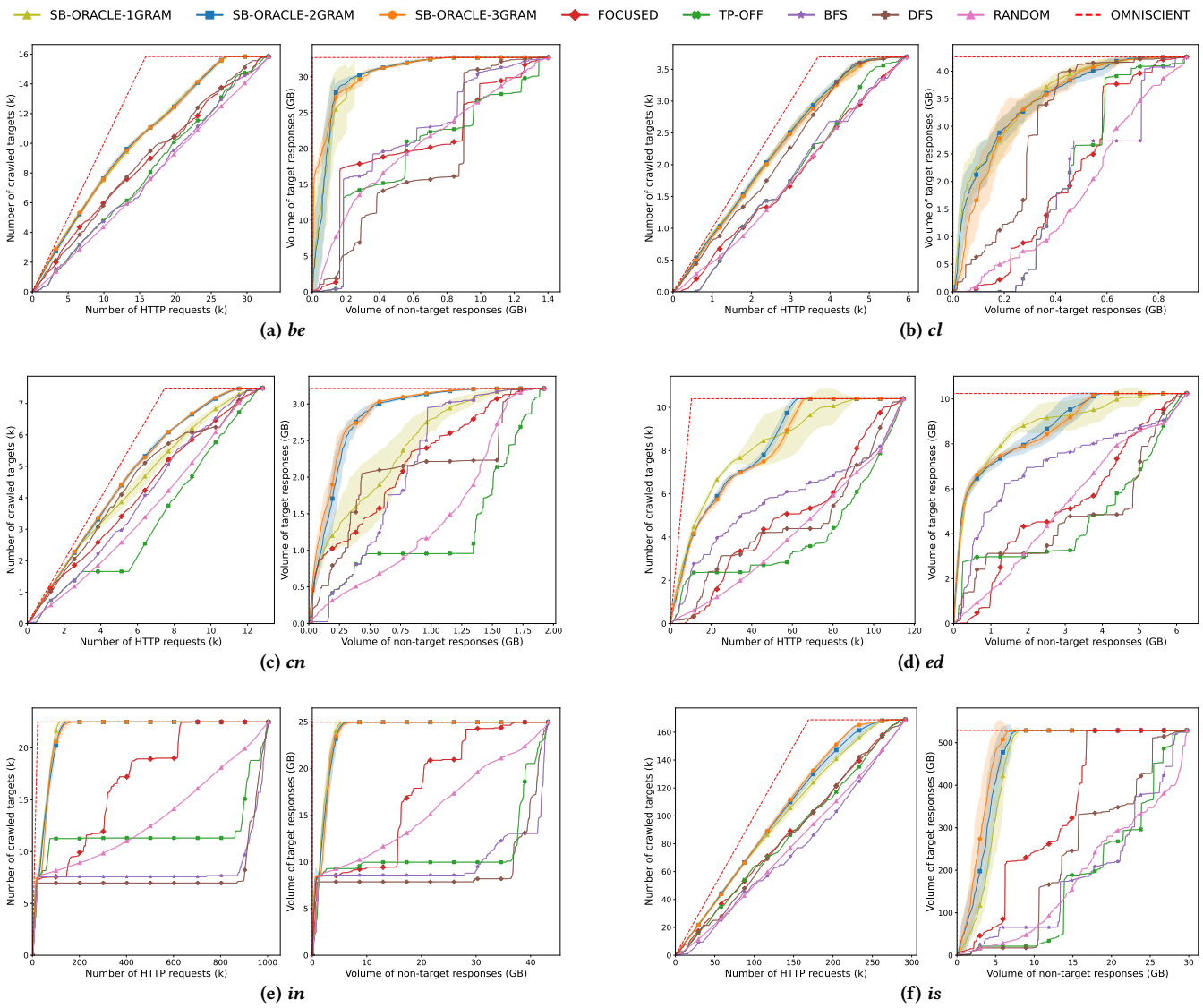


Figure 10: Crawler performance for impact study of the choice of n in n-grams used in tag path vector representation, for websites *be*, *cl*, *cn*, *ed*, *in*, and *is*

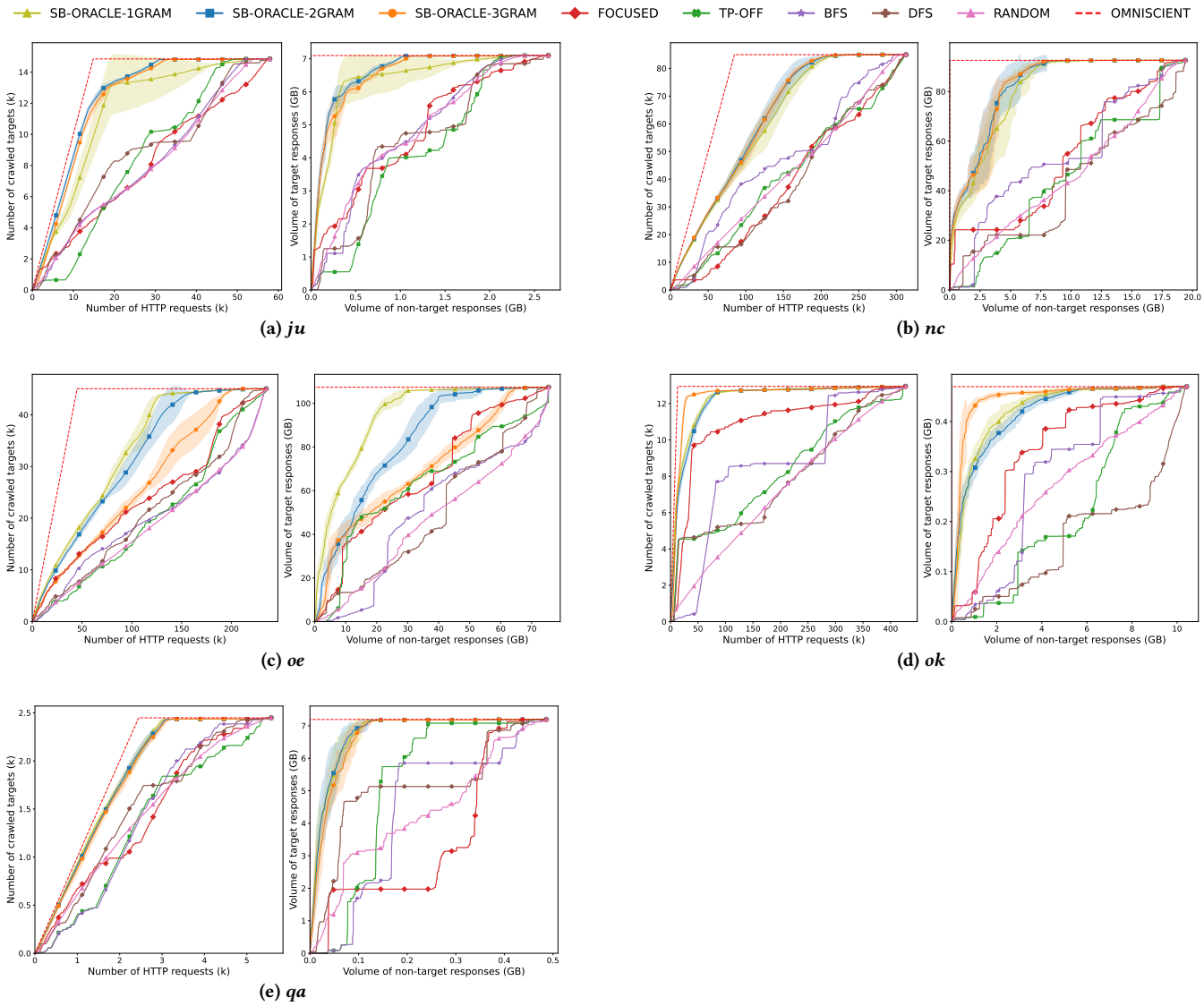


Figure 11: Crawler performance for impact study of the choice of n in n-grams used in tag path vector representation, for websites *ju*, *nc*, *oe*, *ok*, and *qa*

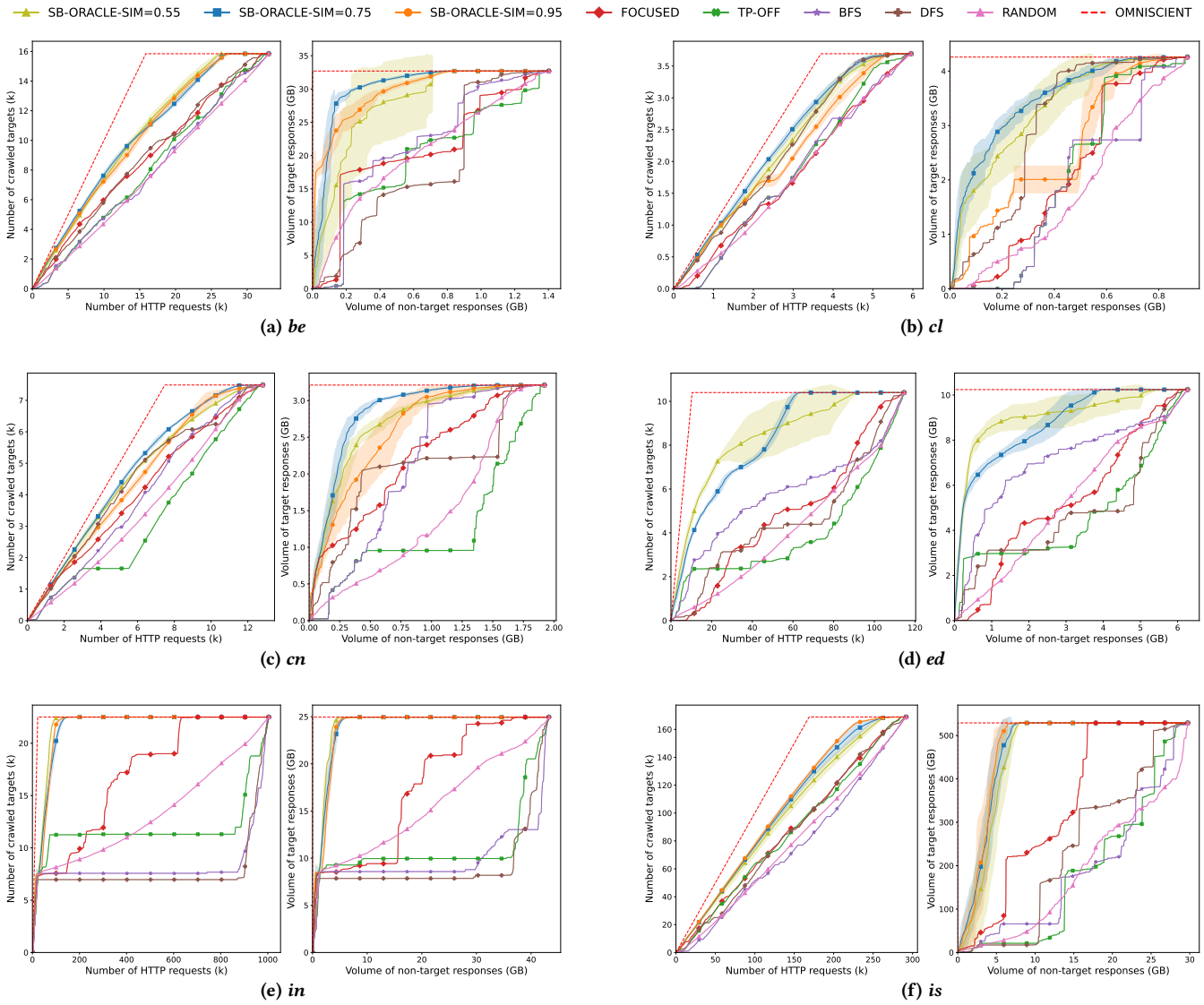
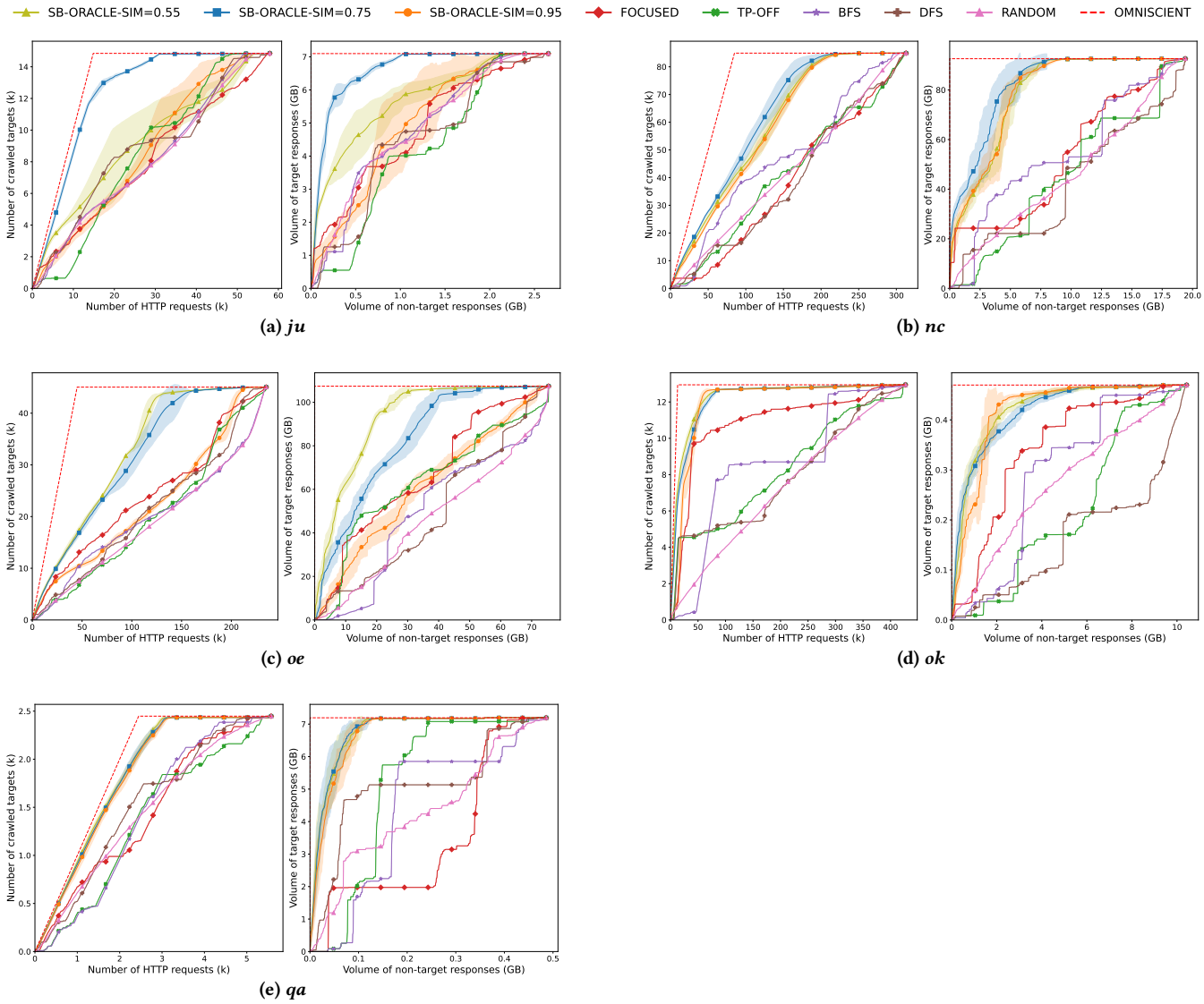


Figure 12: Crawler performance for impact study on similarity threshold θ, *cl*, *cn*, *ed*, *in*, and *is*



(a) *ju*

(b) *nc*

(c) *oe*

(d) *ok*

(e) *qa*

Figure 13: Crawler performance for impact study on similarity threshold θ, for websites *ju*, *nc*, *oe*, *ok*, and *qa*

B.4 Effectiveness of SB Learning

A typical tag path containing reference to IAP in the website *nc* is:

```
/html/body/div.nces/div.container.w-iap/div.iap-content/div.row/div.col-md-6/h4/a
```

and to SIEF in the website *wo*:

```
/html/body/div.wp-page-body.container.default-wrapperpage-collections.collections-sief/div.body-content-
wrap.theme-nada-2/div.about-collection/div.repository-container/div.body/div/p/a
```

Examples of tag paths including keywords related to target retrieval are

```
/html/body.path-node.page-node-type-ressource/div.dialog-off-canvas-main-canvas/div.layout-container/
main#main-content/div.layout-content/div.region.region-content/div.block.block-system.block-system-main-
block#block-open-theme-contenudelapageprincipale/article.mj-resource.mj-break-word.node--promoted/div.
fr-container.mj-ressource__content/div.fr-grid-row.fr-grid-row--gutters.mj-content__corps/div.fr-col-
lg-8.fr-col-offset-lg-2.fr-col-12.content-container__paragraph/section.fr-downloads-group.fr-downloads-
group--multiple-links/ul/li/a.fr-linkfr-link--download
```

for *ju*, and

```
/html/body.home.page-template.page-template-onecolumn-page.page-template-onecolumn-page-php.page.page-
id-7/main.content/div.container/div.row/div.maincol-md-12/article.post.post-7.page.type-page.status-publish.
hentry#post-7/div.entry-content/div#stcpDiv/div/strong/a
```

for *ok*.

Many other websites however have typical target-rich tag paths that cannot be interpreted by humans: for instance on *jp* we have

```
/html/body.top/div#wrapper/div#groval_navi/ul#groval_menu/li.menu-item-has-children/ul.sub-menu/li/a
```

on *il* we have

```
/html/body/div.container.s-lib-side-borders.pad-top-med#s-lg-side-nav-content/div.row/div.col-md-9/div.
s-lg-tab-content/div.tab-pane.active#s-lg-guide-main/div.row.s-lg-row/div.col-md-12#s-lg-col-1/div.s-
lg-col-boxes/div.s-lg-box-wrapper-17054143#s-lg-box-wrapper-17054143/div.s-lib-box-container#s-lg-box-
14451639-container/div.s-lib-box.s-lib-box-std#s-lg-box-14451639/div#s-lg-box-collapse-14451639/div.s-
lib-box-content/div.clearfix#s-lg-content-31285074/ul/li/a
```

and on *ed* we have

```
/html/body.cf-rtm/div.accueil-portail#page/div.container#main-ermes-container/main#main/div#readspeaker-
container/div#portal/div.row/div.col-md-12.cms-inner-layout#layout-2/div.row/div.col-md-6.cms-inner-zone#
zone-4/div#frame-224/div.frame.frame-portalcarouselwebframefactory.hidden-print/div.frame-standard.panel.
panel-front.webframe-ermes-carousel/div.panel-body/div.ermes-frame-html#carousel-4AD04CE72B556A3CCF5DFCF7FB70964E/
div.rsItem/div.modele_13.model-html/table/tbody/tr/td/a.vide
```

B.5 URL Classification Quality

Table 6: Confusion matrix of the URL classifier, on the 11 fully-crawled websites (average over 15 runs)

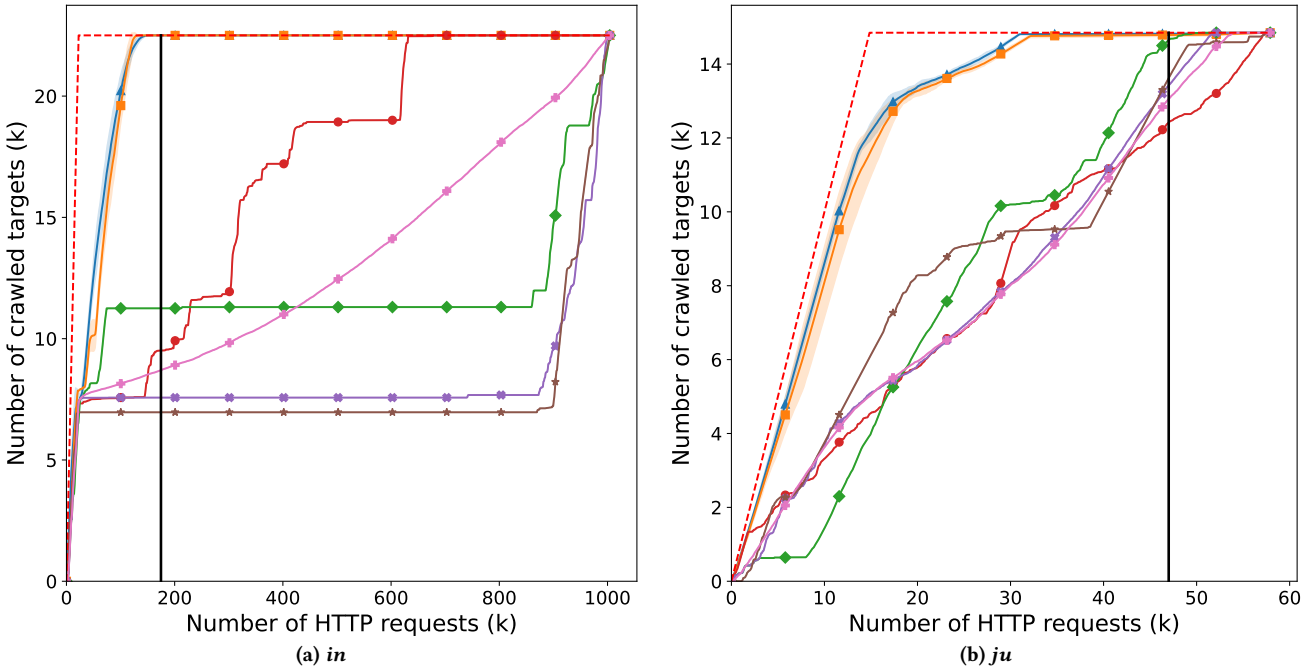
True \ Predicted	HTML (%)	Target (%)	Neither (%)
HTML	58.04	1.37	0.00
Target	0.75	32.19	0.00
Neither	5.34	2.41	0.00

The confusion matrix of our URL classifier, averaged over 15 runs, for all fully-crawled websites, is presented in Table 6.

We observe that **classification errors are extremely marginal on "HTML" and "Target" URLs**. Never classifying as "Neither" leads to some classification errors (discussed in Sec. 3.3), ultimately responsible for the difference between the number of requests made by SB-CLASSIFIER and SB-ORACLE on the 11 fully-crawled websites. However, since the classifier oracle is unfeasible in practice, the performance of SB-CLASSIFIER is satisfactory.

B.6 Early-Stopping

Figure 14 presents a visualization of the early-stopping mechanism applied to two websites: *in* and *ju* (right). Both fall under the first behavior described in Sec. 4.8, illustrating that smaller websites take longer (in %) to stop after the first signs of target convergence. For *in*, stopping occurs shortly after the point where a human would likely have cut, but for *ju*, the delay is more important. Despite early signs of convergence, stopping is postponed due to the fixed $\kappa \cdot \nu = 15\,000$ threshold, which represents a much larger fraction of the site for *ju* (60k pages) than for *in* (1M pages).



(a) *in* (b) *ju*

Figure 14: Visualization of early-stopping time (black rule) for websites *in* and *ju*

Table 7: Characteristics of focused crawler works (LI = Language-Independent; Code = Code publicly available)

System	Focus	Learning Model	States	Actions	Scope	LI	Code
Gouriten et al. [GMS14]	Topic	Bandit	—	URL selection	Across sites	✗	✗
Anthelion [MMB14]	Semantic annot.	Bandit	—	Site selection	Across sites	✓	✓
ACEBot [FS15]	Text content	Offline	—	URL selection	Site-by-site	✓	✗
Han et al. [HWS18]	Topic	MDP	Relevance features	URL selection	Across sites	✗	✗
DSDD [ZSF21]	Topic	Bandit	—	URL selection or SE use	Across sites	✗	✗
TRES [KKP+21]	Topic	MDP	Relevance features	URL selection	Across sites	✗	✓
FOCUSED	Targets	Online	—	URL selection	Site-by-site	✗	✓
SB-CLASSIFIER	Targets	Sleeping Bandit	Available tag paths	URL selection	Site-by-site	✓	✓

C Supplementary material for Section 5 (Related Work)

Web crawlers. Table 7 presents the main characteristics of focused crawler works. We then discuss other relevant literature on Web crawling apart from focused and incremental crawling.

(1) *Distributed* or *parallel* Web crawlers use multiple crawlers concurrently to speed up page retrieval. Key challenges lie in resource (especially, network) management. [CG02] introduced general architectures for parallel crawling, while [BCSV04] proposed UbiCrawler, a decentralized and distributed system using consistent hashing for domain partitioning. Other works [CPWF07, BOM+12] focus on social network crawling. These works aim to efficiently crawl a given set of pages, whereas we focus on minimizing that set. The two could be combined to improve crawling speed and reduce network load.

(2) *Web crawlers targeting specific website types* (such as forums, blogs, or CMS) are built to leverage specific and common structural patterns within these sites. For instance, [GLZZ06, CYL+08] are focusing on forums. While effective, these crawlers are less general, relying on assumptions about site structure. In contrast, our approach adapts to each website dynamically, and needs no assumptions (see Sec. 2.1).

(3) *Hidden-* or *deep*-Web crawlers explore content not reachable via standard hyperlinks, often requiring user interactions such as filling out forms. A comparative study of state-of-the-art deep-Web crawler appears in [HRR19]. In contrast, our work focuses on acquiring specific targets from official websites. In our context, form-based interfaces serve mainly as filters, which are not useful for large-scale retrieval. Moreover, on many institutional websites (see Sec. 4.1), such data is accessible through standard navigation, not portals. Extending our approach to deep-Web crawling remains a natural direction for future work (see Sec. 6).

MAB algorithms. While UCB [ACBF02] and its variants are the state of the art for solving MAB problems, simpler approaches like ϵ-greedy [SB98] randomly select actions with probability ϵ and otherwise choose the highest scoring action. We also considered Bayesian

Bandits, especially TS, a probabilistic method based on posterior distribution estimation. We excluded those approaches to ensure our crawler's *stability*, i.e., ability to give the same output if run several times independently of how the site can change from one crawl to another. Also, Bayesian approaches require prior distributions, unavailable to us due to lack of prior knowledge about the websites. We also did not adopt *linear* or *kernel*-based bandit methods, despite their generalization capabilities. These approaches require stable, qualitative feature representations, extracted from webpages, which are difficult to define in our context. Indeed, while we hypothesize that links appearing in similar tag paths lead to similar content, this assumption does not guarantee the existence of consistent features extracted from the *content* (not structure) of the pages. We would also need to choose *universal* features, i.e., computable for any page of any website we want our crawler to visit. Moreover, most feature-based methods are language-dependent, which conflicts with our aim of being language-independent. Given these limitations, and the good performance we already observed, we focused on AUER [KNS08], the sleeping variant of UCB.

References for the Appendix

[ACBF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, 47(2):235–256, 2002.

[BCSV04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: a scalable fully distributed web crawler. *Softw. Pract. Exp.*, 34(8):711–726, 2004.

[BOM+12] Matko Bosnjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and Luís Sarmento. Twitterecho: a distributed focused crawler to support open research with twitter data. In Alain Mille, Fabien Gandon, Jacques Misselis, Michael Rabinovich, and Steffen Staab, editors, *Proceedings of the 21st World Wide Web Conference, WWW 2012, Lyon, France, April 16-20, 2012 (Companion Volume)*, pages 1233–1240. ACM, 2012.

[CG02] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In David Lassner, David De Roure, and Arun Iyengar, editors, *Proceedings of the Eleventh International World Wide Web Conference, WWW 2002, May 7-11, 2002, Honolulu, Hawaii, USA*, pages 124–135. ACM, 2002.

[CPWF07] Duen Horng Chau, Shashank Pandit, Samuel Wang, and Christos Faloutsos. Parallel crawling for online social networks. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 1283–1284. ACM, 2007.

[CYL+08] Rui Cai, Jiang-Ming Yang, Wei Lai, Yida Wang, and Lei Zhang. irobot: an intelligent crawler for web forums. In Jinpeng Huai, Robin Chen, Hsiao-Wuen Hon, Yunhao Liu, Wei-Ying Ma, Andrew Tomkins, and Xiaodong Zhang, editors, *Proceedings of the 17th International Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, 2008*, pages 447–456. ACM, 2008.

[FS15] Muhammad Faheem and Pierre Senellart. Adaptive web crawling through structure-based link classification. In Robert B. Allen, Jane Hunter, and Marcia Lei Zeng, editors, *Digital Libraries: Providing Quality Information - 17th International Conference on Asia-Pacific Digital Libraries, ICADL 2015, Seoul, Korea, December 9-12, 2015, Proceedings*, volume 9469 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2015.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GLZZ06] Yan Guo, Kui Li, Kai Zhang, and Gang Zhang. Board forum crawling: A web crawling method for web forum. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006), 18-22 December 2006, Hong Kong, China*, pages 745–748. IEEE Computer Society, 2006.

[GMS14] Georges Gouriten, Silviu Maniu, and Pierre Senellart. Scalable, generic, and adaptive systems for focused crawling. In Leo Ferres, Gustavo Rossi, Virgílio A. F. Almeida, and Eelco Herder, editors, *25th ACM Conference on Hypertext and Social Media, HT '14, Santiago, Chile, September 1-4, 2014*, pages 35–45. ACM, 2014.

[HRR19] Inma Hernández, Carlos R. Rivero, and David Ruiz. Deep web crawling: a survey. *World Wide Web*, 22(4):1577–1610, 2019.

[HWS18] Miyoung Han, Pierre-Henri Wuillemin, and Pierre Senellart. Focused crawling through reinforcement learning. In Tommi Mikkonen, Ralf Klamma, and Juan Hernández, editors, *Web Engineering - 18th International Conference, ICWE 2018, Cáceres, Spain, June 5-8, 2018, Proceedings*, volume 10845 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2018.

[KKP+21] Andreas Kontogiannis, Dimitrios Kelesis, Vasilis Pollatos, Georgios Paliouras, and George Giannakopoulos. Tree-based focused web crawling with reinforcement learning. *CoRR*, abs/2112.07620, 2021.

[KNS08] Robert D. Kleinberg, Alexandru Niculescu-Mizil, and Yogeshwer Sharma. Regret bounds for sleeping experts and bandits. In Rocco A. Servedio and Tong Zhang, editors, *21st Annual Conference on Learning Theory - COLT 2008, Helsinki, Finland, July 9-12, 2008*, pages 425–436. Omnipress, 2008.

[MMB14] Robert Meusel, Peter Mika, and Roi Blanco. Focused crawling for structured data. In Jianzhong Li, Xiaoyang Sean Wang, Minos N. Garofalakis, Ian Soboroff, Torsten Suel, and Min Wang, editors, *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 1039–1048. ACM, 2014.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[WW16] Dimitri Watel and Marc-Antoine Weisser. A practical greedy approximation for the directed steiner tree problem. *J. Comb. Optim.*, 32(4):1327–1370, 2016.

[ZSF21] Haoxiang Zhang, Aécio S. R. Santos, and Juliana Freire. DSDD: domain-specific dataset discovery on the web. In Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 2527–2536. ACM, 2021.