Efficient Crawling for Scalable Web Data Acquisition

Antoine Gauquier
antoine.gauquier@ens.psl.eu
DI ENS, ENS, CNRS, PSL, 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, 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. We propose SB-CLASSIFIER, a crawler that 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.

Keywords

Web and Focused Crawling, Data Acquisition, Reinforcement Learning, Scalability, Algorithm

1 Introduction

Openly accessible 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., an album, and describes its attributes such as artist or year. Entity-centric tables have been leveraged to extract large-scale, open knowledge bases [4, 38, 53, 57], and for question answering [13, 30, 31, 56]. They are also frequent in *data lakes*, [12, 18, 21]. In this work, we focus on *openly accessible* Web data, independently of their licensing status. Among them, *statistics datasets* (**SDs**, in short) form a distinct and highly valuable category. They are compiled by domain specialists typically working for national or international governing bodies, such as the United Nations, the IMF, 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 [47], 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 the paper shows, 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%) [43]. 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, 33, 45] in the Information Retrieval and Data Management, and [55] 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 for fact-checking, etc. For example, a work historian may want to retrieve all SDs published by the 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 to retrieve all their SDs, e.g., Eurostat, 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 1 second) between two successive HTTP requests; for a site of 1 million pages, such waits, alone, take 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 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 extensively studied [10, 19, 20, 27, 29, 37, 40]. However, existing approaches often rely on *assumptions that may not hold in our setting*, e.g., that all pages on a topic tend to be close within a website [17]. Early work includes generic focused crawling approaches [10, 19] (that we will illustrate with a FOCUSED baseline); other study large-scale website selection [40] (as opposed to SDs within a given website), topic-driven page retrieval methods such as TRES [37] or [10, 27, 29], and non-topical objectives such as textual diversity [20] (adapted to our setting as a TP-OFF baseline).

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

EDBT '26, Tampere (Finland)

10.48786/edbt.2026.30

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 SB-CLASSIFIER based on reinforcement learning (RL) in Sec. 3, relying on two crucial hypotheses: (i) *links found on similar* tag paths *within HTML web pages* (a tag path [22, 41] 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). SB-CLASSIFIER outperforms all baselines. 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. The code to reproduce the experiments is available in a public GitHub [24], and an extended version of this paper is available in [26].

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,* $\omega(T)$*, 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*, the graph crawling problem 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 $\{u \in V \setminus V' \mid (v, u) \in E, v \in V'\}$: 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 out links in* G.

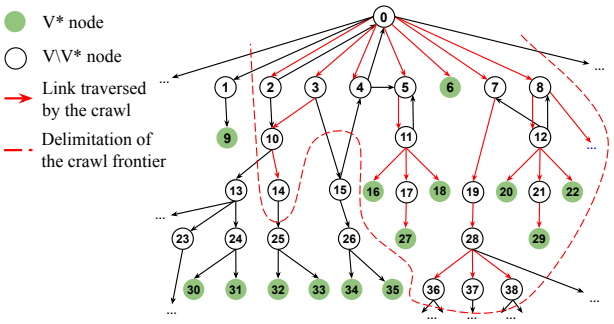

Figure 1: Sample website, crawl, and frontier

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 [26].

Optimal methods being out of reach (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, 48], we use a pragmatic approach. A URL is considered to be part of the same website as the root r 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. 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>`, `<area>`,`<iframe>`.

Since our goal is to find SDs, target pages are those whose *Multipurpose Internet Mail Extensions* (MIME) type is in a *user-defined* list (e.g., `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 is 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 [58], the standard tree-based representation of an HTML page. The tag path includes the **full path of HTML tags from the root 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

[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.

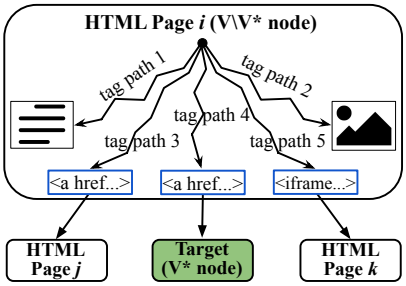

Figure 2: Tag paths in an HTML page

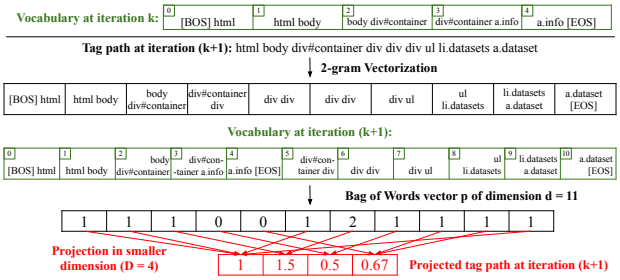

Figure 3: Mapping of a tag path into a fixed-size vector.

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 HTML pages). 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 (much smaller than $\omega(u)$). Sec. 3.3 discusses a MIME type prediction method, resulting in a small, amortized c cost, which we can view as constant.

3 Crawling based on Reinforcement Learning

Our goal in efficient crawling is to retrieve as many targets as possible at minimal cost, without prior knowledge of the website's size or structure. We hypothesize that *an edge's label* λ (its tag path in the page containing the hyperlink) *can be used to learn which edges leads to target-rich pages*. Obtaining and leveraging such insignhts require both *exploration* (to find promising paths) and *exploitation* (to retrieve targets), which we achieve via *reinforcement learning* (RL) [52]. Below, we present the environment (states and actions) of our crawling task (Sec. 3.1), and the learning algorithm (or *agent*) that evolves in it (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) [28]. For our crawler, one might consider using "the currently crawled webpage" as *state*. However, this is not suitable: the current webpage does not reflect previous choices, as it may be reachable via multiple paths. Also, an efficient crawler should not visit any webpage twice, whereas 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.

Each link is labeled with its tag path, so 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 for tag paths encountered at different times have different lengths. To still compute distances between them, 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., $i_1 \neq i_2$ with $h(i_1) = h(i_2)$) are resolved by setting $p_D[h(i_1) = h(i_2)]$ to the *mean* of all elements of p, at positions which collide with i_1, i_2. Positions $0 \leq j < D$ not hit by any i ($h(i) \neq j$ for all $0 \leq i < d$) are set to $p_D[j] = 0$. 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 $D = 2^m = 4, w = 11$ 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 $d_{k+1} = 11$ elements. From it, we compute the d_{k+1}-dimensional BoW vector p_{k+1}. The hash function h maps each input dimension $i \in \{0, \ldots, d_{k+1}-1\}$ to $\{0, \ldots, D-1\}$, e.g., $h(2) = \left\lfloor \frac{(766\,245\,317 \times 2) \mod 2048}{512} \right\rfloor = 1$.

Collisions can occur, e.g., $h(4)=h(8)=h(9)=3$, so $p_{Dk+1}[3] = \frac{p_{k+1}[4]+p_{k+1}[8]+p_{k+1}[9]}{3} \approx 0.67$. This yields a 4-dimensional p_{Dk+1}.

Our next task is to find, for each projected tag path p_D, the nearest action a_c among all known actions A. Conceptually, an action is an evolving cluster of similar tag paths with a centroid: the

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

mean of all projected tag paths. For efficiency, we only store the centroid, which updates 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, it creates a new action. Action centroids are stored in a *Hierarchical Navigable Small Worlds* (HNSW) [39] index $\mathcal{I}$, chosen for its highly efficient updates of centroids as new tag paths join. While maintaining and querying this index might be costly in CPU time, it is negligible compared to crawl time (waiting between two requests, or webpages download). The index also estimates the *cosine similarity* between p_D and its closest centroid: if 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 θ significantly impacts the agent performance. In extreme cases, $\theta = 0$ groups all tag paths into a single action, preventing learning as the agent selects paths randomly, whereas $\theta = 1$ creates a separate action for each path, so the agent only explores and never exploits. A useful threshold balances enough actions to separate interesting from uninteresting paths while still enabling learning and exploitation.

The family of reinforcement models for single-state environments is called *Multi-Armed Bandits* (MABs) [52]. The standard, deterministic, MAB algorithm optimizing the exploration–exploitation trade-off is *Upper-Confidence Bound* (UCB) [3], which implements *optimism under the face of uncertainty*: at each step, a score is computed for each action, and the highest-scoring action is selected.

The MAB score of an action a at time $t+1$, denoted $s^{t+1}_{\text{MAB}}(a)$, combines an exploitation term (the mean of rewards obtained so far for a), and an exploration term, reflecting how often a has been selected relative to the total crawl history: $s^{t+1}_{\text{MAB}}(a) = R^t_a + \alpha\sqrt{\frac{\log(t)}{N_t(a)+\epsilon}}$, where R^t_a is the mean reward for action a up to time t, α weights exploration vs. exploitation, $N_t(a)$ counts how many times a has been selected, and $\epsilon > 0$ prevents division by zero 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) [34] adapts UCB to cope with actions becoming 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 $\mathbb{1}_a(t) = 1$ if action a has remaining unvisited links at time t, and 0 otherwise. After selecting a, our crawler *randomly chooses an unvisited link $l \in a$ to traverse next with equal probability*. How should we reward the crawling along $l \in a$? As 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 encourages the agent to focus on unseen target. For instance, if h has 12 links, 5 leading to targets, and 2 already retrieved, the reward is 3. This simple choice performs well, as shown in Sec. 4.

Yet, a challenge remains: we must compute the reward of all links in h before following any, since each traversal incurs a crawling cost. We therefore need a fast *estimation*, applicable to *any* link, even if some are never followed. For this, we introduce an *online URL classifier* that estimates rewards by inspecting links in the newly acquired page h. We detail it in Sec. 3.3.

Last, we discuss the parameter α in our learning agent. UCB and AUER are optimal for $\alpha = 2\sqrt{2}$ [3, 34], but only under standard conditions where the rewards are normally distributed in $[0, 1]$. In our case, rewards (counts of new targets) are unbounded and typically do not follow a normal distribution (see Sec. 4.7), which is desirable: otherwise, it would mean that most 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 reward distribution, which is not possible. We therefore 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. Doing HTTP HEAD requests is costly and requires spacing for crawling ethics, while using the URL extension (e.g., `.html`, `.csv`) fails for extensionless links 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". HTML pages are added to the crawl frontier, while a target contributes to the crawler's reward. The "Neither" class includes non-HTML pages, URLs with non-target MIME types, or URLs lacking a MIME type from the server. In our experience, over 92% of "Neither" 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 leading to 4xx or 5xx errors from those that resolve fine, Intuitively, as these are often very similar to accessible "HTML" or "Target" URLs. However, *different classification errors have different impacts on crawling*: (1) Misclassifying a "Neither" URL as "HTML" or "Target" only incurs the moderate cost of following it (later, respectively, immediately), as it will likely throw an error. To reduce such visits, we apply 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"), despite knowing some URLs are neither.

[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).

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 $\mathscr{C}(U_{\text{vec}})$

Algorithm 2 details the training and usage of our classifier $\mathscr{C}$. It is a logistic regression model [32], iteratively trained through epochs of *Stochastic Gradient Descent* (SGD) [8], with batch size b. It takes as input a vector of character-wise 2-grams of the 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] an ID, and encode each URL as a BoW over this vocabulary. The choice of model and features is discussed in Sec. 4.6.

As Algorithm 2 shows, in the first training epoch, we label b URLs via HTTP HEAD requests, with X the URL set and y their labels. Based on this, we compute rewards and assign URLs to the frontier $\mathscr{F}$ or targets V^*. After this epoch, we set 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, each HTTP GET issued by the crawler contributes an annotated (URL, class) pair, added to X and y for incremental training when the batch size b is reached. In this way, after the first epoch, we get labels at no extra cost.

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 (Algorithm 3). Initially, the tree T of crawled pages, the set of actions A, and the frontier $\mathscr{F}$ are empty. The budget β spent by the crawler and the crawling step t are set to 0. The crawler starts by visiting the original link r, continuing until all links are visited (the website is completely crawled) or the maximum budget B is reached. At each step, if A is non-empty, the learning agent chooses an action a_c following the Sleeping-Bandit (SB) procedure (Sec. 3.2). and chooses a link from $\mathscr{F}$ associated with a_c uniformly at random. Then, it crawls the page behind this link with Algorithm 4, and if its MIME type matches a predefined blocklist during download, its retrieval is immediately interrupted; this is typically used to avoid multimedia content.

[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** $R_{\text{mean}}(a_c) \leftarrow R_{\text{mean}}(a_c) + \frac{\text{reward} - R_{\text{mean}}(a_c)}{N_t(a_c)}$;

Algorithm 4 starts by retrieving the HTTP GET response: status, MIME type from the header, and body (content) of the webpage. The request cost is added to the budget β. The 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 uncrawled, the crawler follows and processes it. (3) A **2xx** status means the server successfully responds with either an HTML page, a target (whose MIME type is in the list L of 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, keeping only

in-website links. Each new link not in $\mathscr{F}$ or T is classified using Algorithm 2, except when its extension matches a predefined blocklist established before the crawl.

If it points to an HTML page, it is mapped to an action using Algorithm 1. The set A is updated by either *changing an action's centroid* or *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 a_c is updated to reflect the last computed reward.

4 Experimental Results

We now present our experimental results: websites used (Sec. 4.1), poor search engine performance for this task (Sec. 4.2), baselines (Sec. 4.3), crawling setup (Sec. 4.4), comparison of our crawler to baselines (Sec. 4.5), and impact of hyper-parameters (Sec. 4.6). Sec. 4.7 examines our Sleeping Bandit (SB) algorithm's effectiveness in grouping links into coherent and reward-based groups, and Sec. 4.8 presents an early stopping mechanism.

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 SDs (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 in at least two languages have a ✓ in the "**Mlg.**" column. The websites range from a few thousand to over 1M pages, excluding those resulting in 4xx or 5xx HTTP errors (see "**#Available (k)**" column). A ✗ in the "**F. C.**" ("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, and the ratio $\frac{\#\text{Target (k)}}{\#\text{Available (k)}}$ measures *target density*; extremes are 66.78% (*cl*) and 2.49% (*in*). "**HTML to T. (%)**" 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 targets are concentrated in 5.40% of HTML pages). "**Target Size (MB)**" gives the mean and standard deviation (STD) of target file sizes, and "**Target Depth**" the mean and STD of depths (shortest link navigation from the root). *Mean depths vary greatly, from 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, varying in depth, target number and locations, and over 20 languages. This diversity demands *adaptable crawling strategies* to efficiently retrieve targets. A manual analysis of typical target locations on *ju*, *il*, *wh*, and *nc* is provided in the extended version in [26].

4.2 Search Engines and Dataset Search

We attempted to retrieve SDs using search engines (SEs), in particular by testing three popular Google services: classical search (GS), Google Datasets Search (GDS), and the Google Public Data Explorer (GPDE) providing data from international organizations.

GS allows filtering by website (site:) and file type (filetype:), via its Web interface and API. However, SEs, including Google, do not fully index websites and GS caps results at 1k, often truncating them. For instance, querying *ju* for PDFs yields only 302 results, although more than 9k exist; similarly, only 240 ODS files are returned (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 no ZIP archives instead of at least 2 239. GDS trims results even further, e.g., 109 tabular files on *ju* (vs. 1 188); 93 datasets on *il* (vs. ≥170k found by our crawler); and 312 on *ce* (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

We compare our best-performing SB-CLASSIFIER with an adapted version in which the URL classifier is replaced by an (unrealistic) perfect oracle (SB-ORACLE), as well as with several baseline methods. First, we consider four simple crawlers. (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* (OMNISCIENT) 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 closely related focused crawler originally designed to retrieve *diverse* textual content. We apply it *technically unchanged*, except for redefining its reward to target retrieval. As described in [20], it first crawls 3k 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 3k 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 3k pages. Beyond this point, newly formed tag path groups receive a fixed benefit of 0. This baseline, given an unfair advantage in its first stage, can be seen

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

	Starting URL	Mlg.	F. C.	#Available (k)	#Target (k)	HTML to T. (%)	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)

as an *ablated* version of ours, learning *offline* instead of *online* with our RL method.

The *Focused Crawler* (FOCUSED) represents early generic focused crawling approaches that rely on a classifier to estimate the likelihood that a newly discovered hyperlink leads to a target [10, 19]. The frontier is maintained as a *priority queue*, favoring links predicted to be relevant. It relies on a standard *logistic regression*, periodically retrained on crawled pages at no extra HTTP cost. Its features follow standard focused crawler practice: the (approximated) depth of the source page, a 2-gram BoW representation of the URL (similarly to Sec. 3.3), and a 2-gram BoW representation of the link's *anchor* text [10, 19]. *Topic-oriented* features are intentionally excluded (further discussed in Sec. 5). Overall, FOCUSED can be viewed as an *ablated* version of our crawler, as it neither exploits tag-path structure (Sec. 3.3) nor uses RL (Sec. 3.4).

TRES [37] is a topical, RL-based focused crawler using a Bi-LSTM classifier to assess HTML page relevance. Like our approach, it employs RL to learn where interesting content resides in a website. However, TRES defines interestingness through a user-specified topic, whereas we target SDs regardless of subject. As it targets topic-relevant HTML pages only, it requires input keywords, positive HTML examples, and is limited to one language. To include TRES as a baseline for SD retrieval, we made changes related to its focus, *without altering its core logic or technical components*. Specifically, we give it three *unfair advantages*: (i) TRES expects a list of topic-specific keywords to initialize its classifier's training: we supply 74 hand-crafted terms likely to appear in links' anchors to targets to initialize its classifier. (ii) We provide 1k HTML pages with links to targets from the evaluation websites as positive examples (based on prior crawls), as required to pre-train its classifier, giving it partial access to the solution. (iii) TRES must classify URLs as HTML or not to manage its frontier, as it only accepts HTML pages. Although costly in practice (see Sec. 3.3), we gave it an oracle doing this classification at no cost, giving it its chance while not affecting its original behavior. Since its RL agent and classifier rely only on textual HTML features, TRES cannot directly detect targets. We therefore added two adjustments: links are added to the frontier as in [37], while others (which TRES would normally ignore) are visited immediately and targets counted if found; and its language filter was disabled, preventing coarse URL-based false exclusions. The modified code is available at [25].

4.4 Practical Crawling in our Experiments

To evaluate six baselines and our crawler with various hyper-parameters, repeated crawls per website are infeasible due to time constraints (e.g., large files or slow connections), and crawling ethics. Thus, for each website, all baselines and our crawler run in a setting where *each 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 URL, HTTP status, headers, and response body are stored.

For evaluation purposes only, we stop crawling a website under any of the following conditions: (i) One crawler terminates before reaching 1M pages, yielding a complete local replication: all others then rely solely on database look-ups. (ii) All crawlers reach 1M visited webpages (possibly not the same pages across crawlers). (iii) A 3-week time limit is reached, as crawling can be extremely slow for some websites (across 18 sites, 22.2M distinct pages were retrieved). (iv) A crawler exceeds 1 minute between two requests, excluding network latency (only affecting 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*. Full replication enables parallel runs under multiple parameter settings and provides complete knowledge for both (unrealistic) SB-ORACLE crawler and TRES baseline (see Sec. 4.5). (ii) On websites where *all baselines reach 1M webpages*, crawlers are compared on the subset of the website each of them visited. Note that their target sets, and the retrieved data volume, 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 crawl size achieved: e.g., if three crawlers visit 400k, 200k, and 800k pages, respectively, we compare them on their first 200k pages. (iv) On websites where *at*

least one baseline fails to reach 1M pages within the time limit, crawlers are compared on the smallest crawl size achieved: e.g., if crawlers visit 400k, 200k, and 800k pages, results are computed 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 ethics waiting time. To illustrate retrieval times, on the medium-sized website *ed*, SB-CLASSIFIER requires 3h 16min to collect 5k targets and 10h 52min to collect 10k, whereas BFS requires 5h 13min and 48h 45min respectively (1.6× and 4.5× more).

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 [26]. 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), usually large in volume. Full lists are 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. If a crawler never reaches the 90%, we show $+\infty$. The best-performing crawler per website is in bold, excluding SB-ORACLE (as unrealistic). For each website 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 (see Sec. 4.3). This assesses the impact of the URL classifier on crawl performance. Results 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 [24]).

For *partially crawled websites*, we report a single run of SB-CLASSIFIER, lacking perfect classifier information, and since multiple crawls are unfeasible (Sec. 4.4). All baselines but RANDOM and TRES are deterministic, thus one run suffices. TRES is only evaluated on the 10 smallest fully-crawled websites, as it requires oracle access and does not scale: beyond small websites (*be*, *cl*, *cn*, and *qa*), it exceeds the 1-minute-per-request threshold and is stopped before completing the crawl of larger websites. 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 URL classifier outperforms all baselines**, with one exception: on *cl*, DFS is slightly better near the end. On *il* (volume only) BFS briefly surpasses ours, but overall our crawler retrieves twice more targets. **For websites *as, ce, ed, il, in, ju, nc, wh,* and *wo*, SB-CLASSIFIER** significantly reduces resource usage (time or bandwidth) to reach a fixed number of targets or maximize targets under a budget. On *wo*, the best baseline (BFS) would require 3 months to match our targets (linear extrapolation), and FOCUSED 9 months; on *wh*, the best baseline (RANDOM) would require 3 months and the worst (TP-OFF) 6 years. On *il*, the top-3 baselines find no targets in the last 650k iterations, whereas ours continues until the end.

On other sites (*be*, *cn*, *cl*, *jp*, *qa*), while some baselines approach SB-CLASSIFIER performance, it remains superior overall, outperforming all 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.

TP-OFF performs poorly, especially on *ed*, *il*, *jp*, *wh*, and *wo*. Learning from just 3k pages proves insufficient on large websites. On *cn*, TP-OFF underperforms BFS after the initial 3k pages, despite only three times more pages remaining. Increasing the training set could help but would approach the simplicity of a standard BFS crawler. FOCUSED is outperformed by ours on all websites, showing early focused-crawlers are too basic to efficiently solve our problem. 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), 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 [26].

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. target volume); full plots are available in [24]. When varying one parameter, others remain at default values (Sec. 4.5). Additional parameters, including the projection dimension D (Sec. 3.2), showed no significant effect.

(1) *Impact of α.* α controls the exploration–exploitation trade-off (Sec. 3.2). We test $\alpha \in \{0.1, 2\sqrt{2}, 30\}$ (Table 4, top), as optimality of $2\sqrt{2}$ is not guaranteed in our non-standard MAB setting. Smaller α generally improves performance, with best results for $\alpha = 2\sqrt{2}$. Large α leads to excessive *exploration*, especially on low or moderate-reward websites, neglecting useful, already discovered actions. (2) *Impact of n in n-grams for tag path representation.* For merging tag paths (Sec. 3.1), we test $n \in \{1, 2, 3\}$ ($n = 1$ represents paths as sets of HTML tags). Table 4 (middle) shows that $n = 2$ and 3 generally outperform $n = 1$, confirming that groups of tag paths provide a better learning basis than sets of tags. We choose $n = 2$, as 3 yields similar performance but incurs higher computational cost due to exponential growth feature space growth. (3) *Impact of θ.* The similarity threshold θ controls tag path grouping (Sec. 3.1). We test $\theta \in \{0.55, 0.75, 0.95\}$ (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: when 0.55 performs better, 0.75

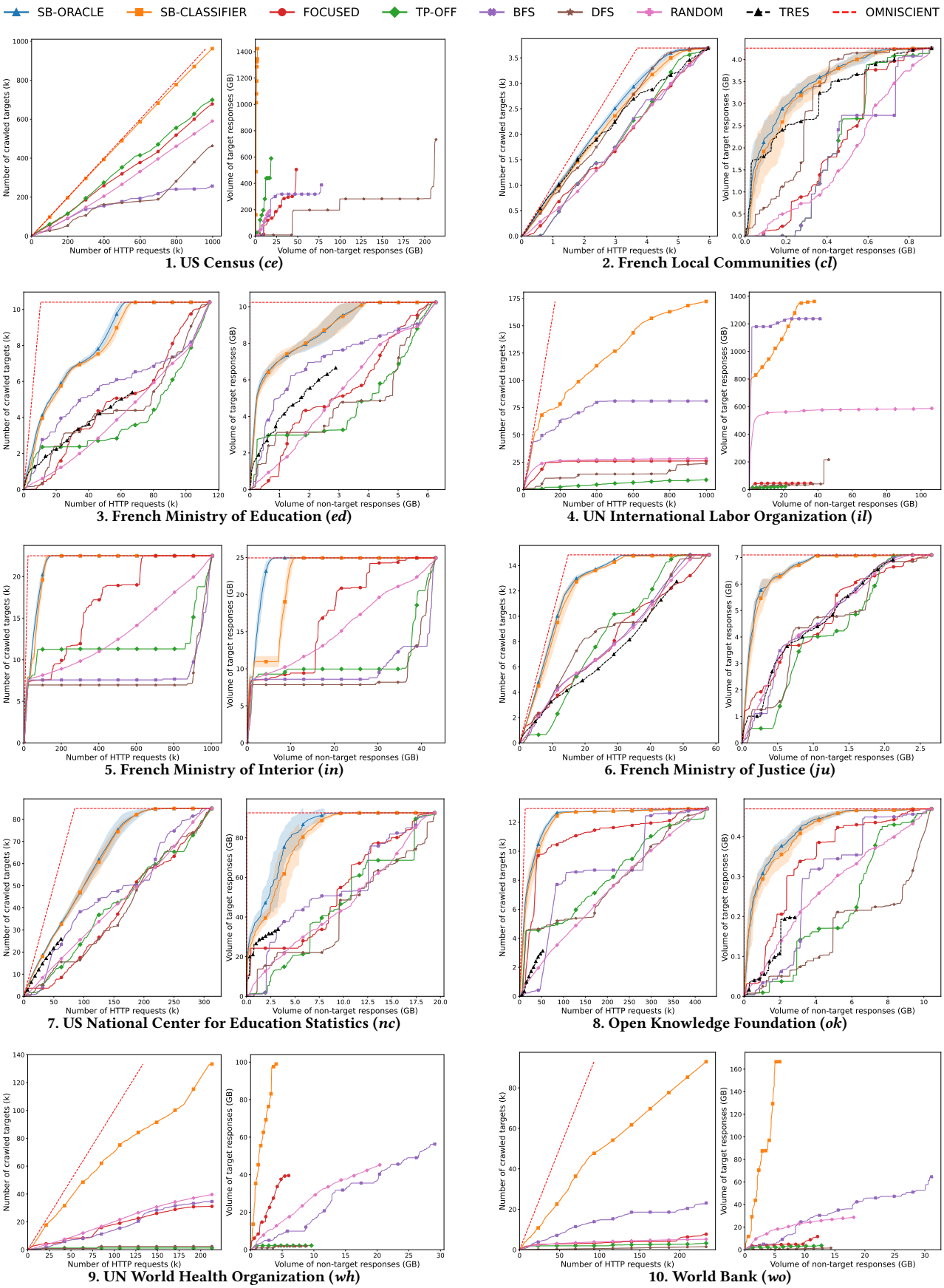

Figure 4: Comparison of different crawler performance for 10 selected websites presented in Table 1; for TRES, experiments are only shown for fully-crawled websites. SB-CLASSIFIER is the proposed approach.

Table 2: Percentage of requests that each crawler performs to retrieve 90% of the targets, for websites in Table 1. Below double horizontal rule, percentage of saved requests and percentage of lost targets due to early-stopping mechanism (see Sec. 4.8).

Crawler \ Website	*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 each crawler retrieves before reaching 90% of total target volume, for websites in Table 1

Crawler \ Website	*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	+∞	+∞

remains comparable, but not conversely (e.g., on *ju*, $\theta = 0.75$ improves both requests and volume by 40%).

All crawls with SB-CLASSIFIER so far use a logistic regression classifier (LR) with URL features (URL_ONLY). To study model and feature impacts, we also evaluate a linear SVM (SVM), multinomial Naive Bayes (NB), and a passive–aggressive classifier [49] (PA), restricting to lightweight online models and excluding deep approaches whose cost would shift the bottleneck from network latency to local CPU/GPU time. Each model is used with two feature sets: URL_ONLY and URL_CONT (URL plus anchor text, DOM path, and surrounding text). All features use character-level 2-gram BoW encoding, yielding eight variants. Experiments are run on fully-crawled websites, with 15 runs per configuration and per site. Table 5 reports, for each website and variant, the *intra-website* crawl metric of Table 2 (number of requests to reach 90% of targets); variants requiring at least 2% fewer requests than URL_ONLY–LR are in bold. The last column (**"MR"**) reports *inter-website* misclassification rates on HTML pages and targets from averaged confusion matrices. On 10 of the 11 websites, no variant improves over URL_ONLY–LR by more than 2%; the sole exception is *qa*, where two variants achieve a $\leq 2.3\%$ gain. This lack of consistent gains, also seen in **"MR"**, shows that neither richer features nor more complex models yield meaningful gains, justifying the choice of URL_ONLY–LR.

4.7 Effectiveness of SB Learning

We first study the reward associated by our algorithm to tag paths. Figure 5 shows the mean reward of the top 10 path groups

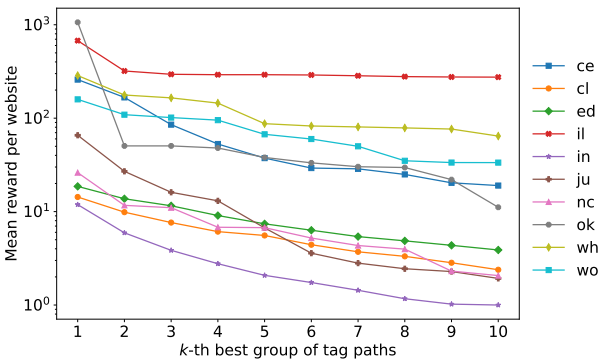

Figure 5: Mean rewards of the top-10 groups of tag paths for the ten selected websites from Figure 4 (log *y*-scale)

(logarithmic *y*-axis), for the ten websites in Figure 4. Top groups have high rewards: across websites, the best group averages 258, followed by 89, 74, 67, and 41 for the 10th, showing that **our SB 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 6, the top 10 groups generally far exceed the overall mean (e.g., for *wo*, the 10th group scores 33.5 v.s. 2.1 for the mean). This is also clear from the high STD values from Table 6

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: Intra-website crawl metric (90% targets reached) and inter-website misclassification rates for URL classifier variants.

Variant	*be*	*cl*	*cn*	*ed*	*in*	*is*	*ju*	*nc*	*oe*	*ok*	*qa*	MR
URL_ONLY-LR	82.1	75.1	71.3	53.2	11.7	76.1	36.5	52.6	60.7	15.9	62.3	2.62
URL_ONLY-SVM	82.7	75.7	71.8	63.6	11.3	76.0	37.4	52.2	63.5	16.7	61.5	2.99
URL_ONLY-NB	82.9	75.2	72.1	53.7	11.4	76.3	35.8	52.7	59.7	18.0	63.1	2.92
URL_ONLY-PA	82.3	74.4	71.7	53.3	11.1	75.8	36.7	51.6	60.5	15.9	60.9	2.56
URL_CONT-LR	82.2	74.4	71.9	54.3	11.3	76.4	37.8	52.9	64.7	16.8	**60.0**	5.93
URL_CONT-SVM	82.6	75.0	71.8	52.8	11.6	76.4	38.8	53.1	61.1	18.7	**60.1**	6.36
URL_CONT-NB	84.1	74.7	71.9	53.6	11.4	75.7	35.5	52.3	59.9	19.1	60.4	7.15
URL_CONT-PA	82.5	75.1	71.9	53.6	11.6	76.2	38.4	52.1	62.6	16.1	60.6	4.12

Table 6: 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

(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 6 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. It further demonstrates how our approach is language-independent: the RL agent learns structural patterns in HTML pages regardless of semantic meaning, enabling efficient target retrieval even on non-English or multilingual websites.

Table 7: SDs retrieval across sample targets

	be	*ed*	*is*	*in*	*nc*	*oe*	*wh*
SD Yield (%)	82	35	93	40	83	60	40
Mean # SDs / Target	9.1	2.8	2.9	2.1	2.1	4.9	1.4

Assessing the concrete precision of our approach to retrieving SDs requires determining the number of SDs collected in the targets. Detecting statistics tables in heterogeneous formats is costly: even state-of-the-art methods require roughly one second per PDF page [51]; spreadsheets pose similar challenges [54]. Performing this online or at scale remains an open problem. As an empirical peak into the SD retrieval precision, we manually analyze a random sample of $7 \times 40 = 280$ targets from 7 diverse websites in Table 7, counting the percentage of targets containing at least one statistics table (**"SD Yield"**) and their mean number per target (**"Mean # SDs / Target"**). Results show that even on non-statistical websites (*ed*, *in*, *oe*, *wh*), an important fraction of targets contains SDs; although many targets contain none, SDs concentrate on a subset, yielding in most cases more than two SDs per collected target.

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 [26] for other relevant work and 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 [27, 29, 37, 59], 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 [40], 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 SD 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 topical locality [17] (similar-topic pages are close) and crawl direction [29] (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 [27, 40, 59] in the form of classical MABs, while some introduce a multiple-state MDP representation as in [29] or [37] which directly extends on [29]. In order to support an MDP representation, [29, 37] 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 prior 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 approaches. Unlike our method, [20] does not use RL but relies on offline training on part of the website (see Sec. 4.3). In most cases, including ours, actions correspond to selecting the next URL to retrieve. Two works differ: [40] applies RL only to select *sites*, to crawl the next URL from, with URL choice handled separately, while [59] additionally allows actions that query a SE. Finally, except for [20], our work differs from all cited approaches in targeting a focused crawl of a single given website: the most common approach crawl across sites without strict scope constraints. This distinction is crucial for our application, which requires retrieving datasets exclusively from trusted sources.

(2) *Incremental* or *revisit policy-based* crawling views a website as an evolving set of pages and selectively revisits them to maximize updated content while minimizing unnecessary requests. [11] presents core challenges, and [50] introduces an incremental version of Heritrix [42], used by the Internet Archive. More recent works propose learning-based revisit policies: [5, 16, 36] predict emergence of new outlinks from page features. Others use RL: [46] shows Thompson Sampling (TS) outperforms MAB alternatives and the Pham Crawler [44], and [35] introduces a learning algorithm to compute optimal revisit policies. While our method is a single-shot crawl, 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 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 Krempf (Radio France) for suggesting the SD retrieval problem. This work was funded in part by the French government under management of Agence Nationale de la Recherche (ANR) as part of the "France 2030" program, reference ANR-23-IACL-0008 (PR[AI]RIE-PSAI) and ANR-23-IACL-0005 (Hi! PARIS Cluster 2030).

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 folder `url_classifier_confusion_matrices` contains all confusions matrices used to compute the "MR" metric in Sec. 4.6 for the URL classifier models. The repository also contains code to reproduce the plots and information about the websites used in our experiments. A second repository [25] contains a fork of the TRES [37] crawler code, adapted for the target retrieval task described in Sec. 4.3. An extended version of this paper is available in [26].

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/AntoineGauquier/tres_modified_for_target_retrieval_non_anonymous.

[26] Antoine Gauquier, Ioana Manolescu, and Pierre Senellart. 2026. Efficient Crawling for Scalable Web Data Acquisition (Extended Version). arXiv:2602.11874 https://arxiv.org/abs/2602.11874

[27] 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

[28] 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

[29] 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

[30] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian 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*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 512–519. doi:10.18653/v1/2021.naacl-main.43

[31] 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

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

[33] 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

[34] 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

[35] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric Joel 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

[36] 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

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

[38] 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

[39] Yury A. Malkov and Dmitry A. Yashunin. 2016. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *ArXiv preprint* abs/1603.09320 (2016). https://arxiv.org/abs/1603.09320

[40] 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

[41] 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

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

[43] 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.

[44] 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

[45] 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

[46] 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

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

[48] 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

[49] Shai Shalev-Shwartz, Koby Crammer, Ofer Dekel, and Yoram Singer. 2003. Online Passive-Aggressive Algorithms. In *Advances in Neural Information Processing Systems 16 [Neural Information Processing Systems, NIPS 2003, December 8-13, 2003, Vancouver and Whistler, British Columbia, Canada]*, Sebastian Thrun, Lawrence K. Saul, and Bernhard Schölkopf (Eds.). MIT Press, 1229–1236. https://proceedings.neurips.cc/paper/2003/hash/4ebd440d99504722d80de606ea8507da-Abstract.html

[50] 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

[51] Marijan Soric, Cécile Gracianne, Ioana Manolescu, and Pierre Senellart. 2025. Benchmarking Table Extraction from Heterogeneous Scientific Extraction Documents. https://arxiv.org/abs/2511.16134

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

[53] 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

[54] Gerardo Vitagliano, Lan Jiang, and Felix Naumann. 2021. Detecting layout templates in complex multiregion files. *Proc. VLDB Endow.* 15, 3 (2021), 646–658.

[55] Andreas Vlachos and Sebastian Riedel. 2015. Identification and Verification of Simple Claims about Statistical Properties. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lluís Màrquez, Chris Callison-Burch, and Jian Su (Eds.). Association for Computational Linguistics, Lisbon, Portugal, 2596–2601. doi:10.18653/v1/D15-1312

[56] 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

[57] 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

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

[59] 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