Efficient Crawling for Scalable Web Data Acquisition

Anonymous Author(s)

Abstract

We consider the problem of efficiently finding all datasets from a Web site, with applications such as data journalism. To this end, we show how to build a Web crawler that retrieves as many *targets*, i.e., resources of certain types, as possible, from a given website, in an efficient and scalable way, i.e., crawling (much) less than the full website. We show that optimally solving this problem is intractable, and propose an approach based on reinforcement learning, namely using sleeping bandits. Our crawler efficiently learns which hyperlinks lead to targets, based on the path leading to the links in their enclosing webpages. Our experiments on websites totaling millions of webpages show that our crawler is highly efficient, delivering high fractions of a site's targets while crawling only a small part.

CCS Concepts

• **Information systems** → **Web crawling**; **Web searching and information discovery**.

Keywords

Web Crawling, Data Acquisition, Reinforcement Learning

ACM Reference Format:

Anonymous Author(s). 2025. Efficient Crawling for Scalable Web Data Acquisition. In . ACM, New York, NY, USA, 36 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Open Data designates datasets whose use is open to all, without access fees or legal restrictions. Open Data has long been shared through the Web; in particular, HTML pages containing tables have been leveraged to extract large-scale, open knowledge bases, e.g. [5, 36, 45, 47]. Such tables typically *describe entities*, such as places, well-known people, commercial and cultural artifacts, etc. *Statistic datasets* (**SDs**, in short) are an interesting, highly useful subset of Open Data. They are compiled by domain specialists working most often for a national government or international organizations, such as the United Nations, the International Monetary Fund, etc. SDs produced by companies may also be published as Open Data, or free for non-profit use. **SD structure strongly differs from entity-oriented Web tables** where each line is about an entity, e.g., album, and describes its attributes, such as artist or year. SDs are mostly **numeric**, e.g., numbers of birth and deaths 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**; they are encoded in a variety of formats, e.g., CSV, TSV, spreadsheets, XML dialects such as SDMX [1]; and sometimes embedded in PDF documents, or serialized in JSON. Some organizations, e.g., Eurostat, publish hundreds of thousands of SDs, updated yearly; others, e.g., a small NGO, may publish a few dozen highly specialized datasets.

SDs are hugely useful: elected officials rely on them to inform decision and support debates; social scientists use them to analyze policy impacts; newsrooms use them to check the accuracy of public statements, make comparisons across countries, etc. Research works seeking to automate the recognition and checking of claims based on SDs include, e.g., [8, 13, 32, 40].

To leverage Open SDs, one first needs to *acquire* them. Certain institutions specialize in providing such SDs and offer exhaustive access through APIs (e.g., Eurostat, Chicago Data Portal). However, many others lack such systems but still host numerous high-quality SDs. For example, a user may ask: **find all datasets published by the International Labor Organization** (ILO). Unlike entity-oriented Web tables, SDs are overwhelmingly not embedded in HTML pages, but published as separate files, which we must find. *Search engines* (**SEs**, in short) and associated data portals turn out to provide access to **only a tiny fraction** of existing SDs (Sec. 4.2); also, how these are selected is opaque to users. A **naïve, exhaustive website crawl is extremely inefficient** for large websites: (i) acquiring a full website takes space and time; (ii) *crawling ethics* requires to wait (typically around 1 second) between two successive HTTP requests to the same server; for a site of 1 million pages, such waits, alone, take more than 11 days. Newsroom usage, for instance, requires a much faster acquisition; also, journalists and scientists lack access to SE-scale infrastructure. 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. Here, *target* is user-defined; motivated by SD retrieval, we describe them to our crawler as *data files (CSV, spreadsheet, etc.)*. However, our work **is compatible with *any* definition of target**.

Focused crawlers have been studied in prior work [14, 24, 25, 27, 38]. Some aim to select some websites among thousands [38], as opposed to SDs within a given website; others aim at webpages on certain predefined topic [14, 24, 25, 27]. As we will show, traditional focused crawlers make different assumptions from this work, e.g., that all pages on a topic tend to be close within a website [22], and estimate page utility in a fundamentally different way, leading them to perform poorly on our problem (Sec. 4.5).

Note that we do not consider crawling the *deep web*, i.e., feeding inputs to forms in order to discover hidden content. Instead, motivated by the large numbers of open, valuable SDs accessible by navigating links, we address the challenging problem of **retrieving SDs in a website as efficiently as possible, with no assumptions about the website structure** (other than the ability to follow links). Note that forms may be scattered across a website. Thus, before a deep web crawl, a crawler like ours needs to find the forms. This is the case of the ILO website, where, furthermore, our crawler found all targets by following links (with no need for forms).

Contributions. Our contributions are as follows. (1) We formalize our **graph crawling problem** and show that optimally solving it is intractable (Sec. 2). (2) We propose a **novel approach based on reinforcement learning (RL)** in Sec. 3. Our approach relies on two crucial hypotheses: (i) *similarly structured hyperlinks lead to similar content*, where by hyperlink structure we denote the path leading to the link, in the HTML page where the link appears; and (ii) *we can*

learn, from the link structure, which links are likely to lead to targets. (3) We demonstrate the effectiveness and efficiency of our approach through **extensive experiments on diverse websites**, on 22.2 millions pages (Sec. 4). Our crawler **outperforms all baselines**, including the closest equivalent from prior focused crawlers [24]. In our experiments, **our crawler retrieves 90% of the targets accessing only 20% of the webpages** in some huge websites.

We discuss related work in Sec. 5. For space reasons, the proof of Proposition 4, additional experiments (including a hyper-parameter study), and a longer discussion on related work can be found in [3], along with open-source code to reproduce the experiments.

2 Problem Statement and Modeling

We formalize our Web data acquisition problem as a *graph crawling problem*, and show it is NP-hard, even if the website is known before the crawl (it is not!). Then we discuss how exactly we map our problem into the graph crawling framework, including a crucial choice of labels for the graph edges.

Graph Crawling Problem. We model a website as a rooted, node-weighted, edge-labeled directed graph. Each node represents a webpage, and each edge a hypertext link. We fix a countable set $\mathcal{L}$ 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* vertices *(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);* $\lambda : E \to \mathcal{L}$ *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 *of* 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 as $\{u \in E \setminus E' \mid (v, u) \in E, v \in E'\}$: the edges whose source node has been crawled, while the target has not. Figure 1 shows a sample 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* $\omega(T) \leqslant B$ *is NP-complete; hardness holds even when* ω *is a constant function.*

This is the decision variant of the graph crawling problem. Hardness is by reduction from the set cover problem [31]. As a consequence, optimal methods being out of reach (interesting websites may have millions of pages), we need *heuristic* methods reaching *low cost in practice.*

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.

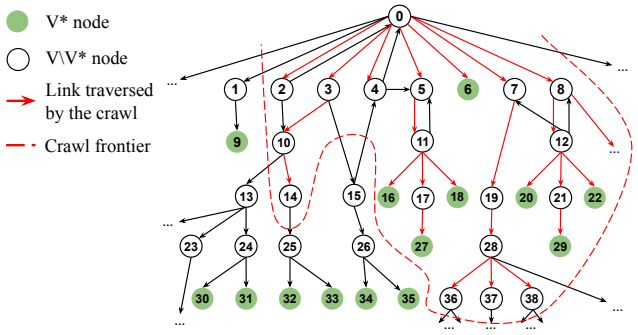


Figure 1: Sample website, crawl, and frontier

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 [2, 42], we use a pragmatic approach. A URL is considered to be part of the same website as the root r, if and only if *its hostname* (the part of the URL which is after the scheme but before the path, with a potential "www." prefix excluded) *is a subdomain of the hostname of* r (a potential "www." prefix also excluded). V contains all such URLs. For instance, if r is https://www.A.B.com/index.php, the URLs https://www.A.B.com/folder/content.php and https://www.C.A.B.com/page.html are part of V, but https://www.B.com/page.php and https://sigir2025.dei.unipd.it/ are not.[1] An edge $(u, v) \in E$ exists if u links to v via HTML tags like `<a href="...">`, `<area>`, `<iframe>`), etc.

For statistic fact-checking applications [7, 13], **target webpages** are those whose *Multipurpose Internet Mail Extensions* (MIME) types are in a *user-defined* list (e.g., `application/csv`, `text/csv`, `application/pdf`, `application/vnd.ms-excel`). Non-target types include `text/html`, `video/*`, `audio/*`, `image/*`, etc. The MIME type can be obtained via HTTP HEAD requests. Depending on application needs, any other target MIME type set can be used.

To each **edge** e, the edge labeling function λ associates a **label** $\lambda(e)$, derived from the HTML page's DOM [48], the standard tree-based representation of a page. The label includes the **full path of HTML tags from the root (the `html` tag) to the hyperlink tag** (`a`, `area`, `iframe`, etc.), along with class and id attributes describing additional structural and styling information. For example, a label might be "`html body div#main ul.datasets li a`", where `#` prefixes the HTML ID, and `.` indicates a class. This labeling makes it very likely that *links labeled with similar DOM paths lead to similar content types*, even across different webpages of a website.

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

[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.

3 Crawling based on Reinforcement Learning

We aim to retrieve as many targets as possible at minimal cost, without prior knowledge of the website's size, structure, or target locations. We hypothesise that *an edge's label* (derived from the DOM path leading to a hyperlink within its page) *can be used to learn which edges leads to target-rich pages*. To leverage such insights, we need to both *explore* the website (find promising paths), and *exploit* the knowledge thus gained (to retrieve targets). Thus, our approach to focused crawling uses *reinforcement learning* (RL) [44]. Below, we present the environment (states and actions) of our crawling task (Sec. 3.1), and the learning algorithm (the *learning agent*) evolving in it (Sec. 3.2). We describe the URL classifier for rewards computation (Sec. 3.3), and the overall crawling algorithm (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 action), that maximizes the total reward. This model is known as a *Markov Decision Process* (MDP) [29].

Let us consider what we could use as *state* in our crawler. One could think of using "the currently crawled webpage", but this is unsuitable: it does not reflect previous choices, as a webpage may be accessible via several navigation paths. Also, efficient crawling should avoid revisiting webpages, whereas learning requires multiple visits to refine and then leverage information learned in that state. Thus, we use a **single-state model**, where the agent is always in one state. What matters then is the set of actions available at each crawl step. Intuitively, an *action* is to navigate along a link.

Grouping Links into Actions. As we hypothesize that similar links have similar values (likelihood of leading to target), we define an action as *a group of similar links* and execute it by crawling along these links. As we *label each link based on the DOM path leading to the link in its enclosing page*, we need a **measure of similarity between paths**. We thus represent each DOM path as a *numerical bag-of-words (BoW) vector* p based on an n-gram vocabulary built from all encountered DOM paths. We use n-grams to preserve the order of HTML node names in paths, as this order is significant (more details in [3]). Since the n-gram vocabulary grows dynamically during the crawl, BoW vectors assigned to links encountered at different times vary in length. To compute distances between these representations, we first *project each 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* $h : x \mapsto \left\lfloor \frac{(\Pi \times x) \mod 2^w}{2^{w-m}} \right\rfloor$ mapping any integer x to a number between 0 and $D-1$. Its parameters Π (a large prime number) and w are fixed, 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 transforms p into a D-dimensional p_D, where for every $0 \leqslant i < d$, $p_D[h(i)] = p[i]$. (3) Potential collisions of h, i.e., $i_1 \neq i_2$ such that $h(i_1)=h(i_2)$, are resolved by assigning to $p_D[h(i_1)=h(i_2)]$ the mean of all elements of p at positions which collide with i_1, i_2. Further, there may be positions j between 0 and D such that for every $0 \leqslant i < d$, $h(i) \neq j$. For such positions j, $p_D[j] = 0$. This happens, e.g., at the beginning of the crawl, when the set of HTML element names seen so far is small (thus d is small). Figure 2 shows the projection of a DOM path via 2-grams, into a 4-dimensional vector (BOS, EOS denote begin/end of string).

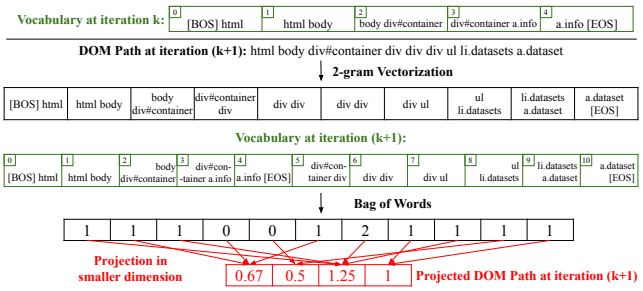


Figure 2: Mapping a DOM path to a vector, with $n = 2$.

Algorithm 1: Finding the action for a hyperlink

Input: Action set A, projected DOM 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 $s_c \geqslant \theta$ **then**
  Update centroid of a_c in $\mathcal{I}$ with respect to p_D;
else
  Add a_{new} to A;
  Add a new entry a_{new} to $\mathcal{I}$, with value p_D;
  $a_c \leftarrow a_{\text{new}}$;
return a_c

Next, we **find the nearest action** a_c to each projected link representation p_D among all known actions A. Conceptually, we see an action as an **evolving set (cluster)** of similar links with a **centroid**, the mean of all projected links. For efficiency, we only store the centroid, which updates as the action's set of links evolves.

Algorithm 1 computes the nearest action of each p_D if it is close enough; otherwise, a new action is created and returned. We insert the action centroids into a *Hierarchical Navigable Small Worlds* (HNSW) [37] index denoted $\mathcal{I}$, chosen for its highly efficient updates when a new link joins an action (centroid update). While maintaining and querying the index might be costly, this cost is negligible compared to the crawl time (waiting between two requests, or for the download of webpages). The index estimates the *cosine similarity* between p_D and its closest centroid. If the similarity is above a **threshold** θ, the link is added to the action. Otherwise, a new action composed of only p_D is created. The choice of the threshold has a significant impact on the performance of our learning agent (see [3]). As an extreme case, if $\theta = 0$, all links are grouped into a single action, and the agent cannot learn anything; it will randomly select the links to follow. On the other hand, if $\theta = 1$, each link makes up a separate action. As a result, the agent only explores, and never exploits. A useful threshold should enable enough actions to separate interesting groups of links from non-interesting ones, while still allowing learning and exploitation.

Concretely, for our crawler, an action a means *navigating along a link from* a. Next, we show how our RL agent builds and exploits knowledge about a website while crawling it.

3.2 Learning Agent: Sleeping Bandit

The family of RL models for stateless (one-state) environments is called *Multi-Armed Bandits* (MABs) [44]. The standard, deterministic, MAB algorithm optimizing the trade-off between exploration and exploitation is *Upper-Confidence Bound* (UCB) [4], based on the principle of *optimism under uncertainty*. At each time step, a score is computed for each action, and the action with the highest score is chosen. The MAB score of action a at time $t+1$, denoted $s^{t+1}_{\text{MAB}}(a)$, includes an **exploitation term** (mean reward for the action at t) and an **exploration term** reflecting the number of times the action has been selected relative to the crawling history length: $s^{t+1}_{\text{MAB}}(a) = R^t_a + \alpha\sqrt{\frac{\log(t)}{N_t(a)+\varepsilon}}$, where R^t_a is the mean reward obtained for action a until time t, α is a parameter weighing exploration vs. exploitation, $N_t(a)$ is the number of times a has been selected up to t, and $\varepsilon > 0$ a small value to avoid division by 0 when $N_t(a) = 0$.

Since each URL is visited only once, an action may become empty after all its URLs are visited. [33] presents an alternative to UCB, the *Awake Upper-Estimated Reward* (AUER), which shows how to compute the reward when some actions become unavailable, or *sleeping*. Following the AUER model, our score becomes: $s^{t+1}_{\text{SB}}(a) = \mathbb{1}_a(t) \times \left(R^t_a + \alpha\sqrt{\frac{\log(t)}{N_t(a)+\varepsilon}}\right)$, where $\mathbb{1}_a(t)$ is 1 if there are remaining links in action a at time t; 0 otherwise. After selecting an action a, our crawler randomly chooses an unvisited link l from a to traverse next, ensuring equal selection probability for all links within a.

How should we reward crawling along $l \in a$? Since our goal is to find *new targets*, the reward should reflect the number of *not-yet-discovered target links* in the HTML page h reached via l. This "novelty" condition helps the learning agent focus on unseen targets. For instance, if h contains 12 links, including 5 leading to targets, with 2 of these already seen, the reward for following l to h would be 3. This simple choice gives good results, as shown in Sec. 4.

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

Last, we discuss the parameter α in our learning agent. UCB and AUER are optimal with $\alpha = 2\sqrt{2}$, but only under standard conditions where rewards are normally distributed in $[0, 1]$. In our case, rewards (counting new targets) are unbounded and unlikely to follow a normal distribution. In fact, we *hope* this is *not* the case: it would mean that a majority of HTML pages contain links to targets. If this held, we would have to visit the entire website, or at least a vast share, to retrieve most targets. Setting α for optimality would require knowing the shape of the distribution of rewards, which is impossible. Thus, we keep $\alpha = 2\sqrt{2}$, which performs well across varied reward distributions (as shown in Sec. 4.1, in Figure 3).

3.3 Online URL Classifier

Determining the reward of an agent action requires assessing a page's MIME type from its URL. While an **HTTP HEAD** request can provide this information, it is costly and requires spacing for crawling ethics, e.g., 1-second intervals. Relying on URL extensions (last URL segment, e.g., `.html`, `.pdf`, `.csv`) fails for links without extensions, such as https://www.justice.gouv.fr/en/node/9961, or on sites like ILO. To be efficient and generic, we devise an **online URL classifier to estimate rewards**, as follows.

Algorithm 2: Online URL classifier procedure

Input: U, a URL
Output: $l_U \in$ {"HTML", "Target"}
if $|X| \geqslant b$ **then**
  $X_{\text{mat}} \leftarrow$ 2-gram bag-of-words of each URL in X;
  C is incrementally trained on batch (X_{mat}, y);
  $(X, y) \leftarrow (\emptyset, \emptyset)$, and 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), and **return** l_U;
else
  $U_{\text{vec}} \leftarrow$ 2-gram BoW vector of U, and **return** C (U_{vec})

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

Initially, our classifier assigned one of three classes to each link found in a newly crawled page h. However, it struggled to distinguish URLs resulting in 4xx or 5xx errors from those that resolve fine. Intuitively, this is because the former URLs often resemble valid (accessible) URLs of "HTML" or "Target" pages. However, *different classification errors impact crawling differently*: (1) Misclassifying a "Neither" URL as "HTML" or "Target" only incurs the moderate cost of following the URL (immediately in the second case, later in the first case), as it will likely throw an error. (2) Misclassifying "HTML" or "Target" as "Neither" *completely withdraws the page from the crawl*: if it is a target, we miss it; if it is HTML, we miss all URLs that might be discovered by following it. This can make the crawler miss huge parts of the website. To avoid the second kind of errors, our classifier is trained on just **two classes ("HTML" and "Target")**, even though some URLs are neither.

Algorithm 2 details the training and usage of our classifier C, a logistic regression model [30] trained via *Stochastic Gradient Descent* (SGD) [10] with batch size b. The classifier takes as input a vector of character-wise 2-grams of the original URL. For example, the URL https://www.A.com/data/file.csv is transformed into a list [ht, tt, tp, . . . , .c, cs, sv]. Each ASCII character[2] pair is assigned a unique ID, encoding the URL as a BoW over this vocabulary.

As Algorithm 2 shows, during the first training epoch, we label b URLs using HTTP HEAD requests; with X representing the set of URLs and y their labels. Based on this, we compute rewards and assign URLs to either the frontier $\mathcal{F}$ or the target set V^*. After this epoch, we set the Boolean initial_training_phase to false and use

[2]ASCII digits, upper and lower case letters and main special characters.

Algorithm 3: Efficient target retrieval

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

the classifier to infer URL classes without additional HTTP HEAD requests. The classifier is further improved through *online training*: **during execution, any HTTP GET request issued by our crawl contributes an annotated (URL, class) pair** which is added to X and y for incremental training when reaching the batch size. In this way, after the first training epoch, we get labels at no extra cost. To quickly deploy and frequently improve the classifier's accuracy, b should remain 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 with different URL structures), unlike an offline method.

3.4 Crawling Algorithm

The overall crawling procedure is described in Algorithm 3. Initially, the tree T, the action set A, and the frontier $\mathcal{F}$ are empty, and both the **budget** β spent by the crawler and the crawling step (or time) t are set to 0. The crawl starts by visiting the original link r, continuing until either all links are visited (the website is completely crawled) or the maximum budget B is reached. At each step, if the set of actions is not empty, the learning agent choses an action a_c following the SB procedure (Sec. 3.2), and then uniformly at random selects a link from the frontier to crawl, using Algorithm 4.

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

For each link, if its URL is not already in $\mathcal{F}$ or in T, it is assigned a class by calling Algorithm 2. If it points to an HTML page, it is mapped to an action using Algorithm 1. The set of actions A is updated, either by changing an action's centroid or adding a new action, and the chosen action a_c is mapped to the link, which is added to the frontier $\mathcal{F}$. If the link leads to a target, it is immediately retrieved by recursively calling Algorithm 2, and the reward is

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 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);
      $U_{\text{new}} \leftarrow$ Internal hyperlinks in body;
    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 $u \notin T \cup \mathcal{F}$ **then** crawl_next_page(u) and **return**;
reward $\leftarrow 0$;
for $u_{\text{new}} \in U_{\text{new}}$ such that $u_{\text{new}} \notin T \cup \mathcal{F}$ **do**
  $l_{u_{\text{new}}} \leftarrow$ 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 $\mathcal{F}$;
  else if $l_{u_{\text{new}}} =$ *"Target"* **then**
    crawl_next_page(u_{new}), and reward $\leftarrow$ reward + 1;
if $u \neq r$ **then** $R_{\text{mean}}(a_c) \leftarrow R_{\text{mean}}(a_c) + \frac{\text{reward} - R_{\text{mean}}(a_c)}{N_t(a_c)}$;

updated. Finally, if the page crawled by current call of Algorithm 2 is not the starting page, the mean reward for the chosen action a_c is updated to reflect the last computed reward.

4 Experimental Results

We 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) and comparison with baselines (Sec. 4.5). Sec. 4.6 evaluates the SB algorithm's effectiveness in grouping links into coherent, reward-based groups, and Sec. 4.7 analyzes the URL classifier's quality. A complete hyper-parameter study is in [3].

4.1 Websites

We experimented on 18 websites with varied characteristics and report graphical results on a representative selection of six (Table 1); full results appear in [3]. The sites are: the Australian Bureau of Statistics (*ab*), the French national assembly (*as*), the US Bureau of Economic Analysis (*be*), the US Census (*ce*), the French local communities (*cl*), the French council for statistical information (*cn*), the French ministries of interior (*in*), education (*ed*) and justice (*ju*), the UN's International Labor Organization (*il*), the French official statistical institute (*is*), Japan's Ministry of Interior (*jp*), the US National Center for Education Statistics (*nc*), the Organization for Economic Co-operation and Development (*oe*), the Open Knowledge Foundation (*ok*), Qatar's official statistical service (*qa*), the UN World Health Organization (*wh*), and the World Bank (*wo*).

Table 1: Main characteristics of six selected websites (same as Figure 3), out of 18. See detailed crawling methodology in Sec. 4.4.

	Starting URL	Mlg.	Fully Crawled	#Available (k)	#Target (k)	HTML to Target (%)	Target Size (MB)	Target Depth
ce	https://www.census.gov/	✗	✗	988.37	257.68	3.47	1.51 (± 15.77)	4.23 (± 0.48)
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)
ju	https://www.justice.gouv.fr/	✗	✓	56.61	14.85	4.85	0.48 (± 1.34)	86.91 (±86.30)
ok	https://okfn.org/	✓	✓	423.12	12.95	0.74	0.04 (± 0.24)	2.64 (± 2.89)
wo	https://www.worldbank.org/	✓	✗	223.67	23.10	2.38	2.80 (± 27.16)	4.52 (± 0.69)

The websites range from a few thousand to a million pages or more, excluding those resulting in 4xx or 5xx HTTP errors, as noted in the "**#Available (k)**" column. A ✗ in the "**Fully Crawled**" column indicates that crawling was stopped before visiting the entire website (see Sec. 4.4). Websites with pages in at least two different languages have a ✓ in the "**Mlg.**" column.

The following metrics assess the difficulty and interest of a target-oriented crawl in a website. For websites not fully crawled, they are computed on a subset of the site using a BFS crawl (Sec. 4.4). "**#Target (k)**" is the total number of identified targets (in thousands). The ratio $\frac{\#\text{Target (k)}}{\#\text{Available (k)}}$ measures the *target density*; extremes are 66.78% in *cl* and 2.49% in *in*. "**HTML to Target (%)**" is the percentage of HTML pages with at least one hyperlink leading to a target, indicating the *density of target-pointing pages* of a website. For instance, *cl* has the highest target density, but these links are concentrated in only 5.40% of the HTML pages. "**Target Size (MB)**" is the average file size of a target, along with its standard deviation. "**Target Depth**" is the mean and standard deviation of target depths, defined by the smallest number of links from root to target. Depths vary greatly, both in mean and variance. Highest average depths arise on portals that require page-to-page navigation, such as *ju*.

Table 1 shows websites are diverse and heterogeneous: small or huge, with varying depths, numbers of targets, depths, target locations, etc.; they use more than 20 languages. Such diverse websites require very different crawling efforts; our crawler must find out how to quickly retrieve targets in each of them.

4.2 Search Engines and Dataset Search

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

GS allows filtering by website (site:) and file type (filetype:) through its web interface or API. However, SEs, including Google, **do not fully index any website**. Google **limits results to 1,000**, leading to drastic truncations. For instance, querying *ju* for PDFs yields 302 results, while 9 000 exist. Similar issues arise for other MIME types and websites: on *ju*, GS lists 240 ODS files (out of 910), and on *in*, 38 XLS (out of 1 546). Even worse, GS also fails to recognize TSV files, missing all 11 097 TSVs on *ju*. On *il*, GS retrieves 641 results (instead of at least 49 962), etc. GDS **trims results even further**, e.g., only 109 tabular files for *ju* (out of 1 188). For *il*, it returns 93 datasets (our crawler finds over 170k) but cannot detect their MIME types. Similarly, it finds just 312 datasets for *ce* out of 800 000, etc.

GPDE is built for human website readers and indexes SDs from a **closed, fixed set of providers**, including Eurostat, *wo*, Canadian Statistical Institute, and *oe*. However, it excludes key sites like UN's *wh* and *il*, and US sites such as *be*, *nc*, and *ce*. GPDE **only allows manual exploration**, plotting datasets guided by users' filters. It offers **no download** facility or pointers to original datasets. While it allows inspecting 150 OECD statistics over 37 countries and 100 years, the only reference is a a 272 pages PDF, in the OECD iLibrary.

Overall, SEs lack **transparency and control over selected SDs**, a critical limitation, especially for data journalism. In contrast, our crawler retrieves all data files within a defined budget, with explainable decisions, and a choice over MIME types to target.

4.3 Baselines

First, we consider four simple baselines. (i) The *random crawler* selects the next hyperlink to visit uniformly at random from the crawl frontier. (ii) *Breadth-First Search (BFS)* uses a *First-In-First-Out queue* for the frontier, crawling all pages at path length ℓ before those at $\ell' > \ell$. (iii) *Depth-First Search (DFS)* uses a *Last-In-First-Out stack* but is rarely used due to susceptibility to robot traps. (iv) The *Omniscient Crawler*, an unrealistic baseline, knows all target URLs (V^*) and crawls them directly, providing an unreachable upper bound on efficiency, as an optimal crawler is intractable (Prop. 4).

The *Offline-Trained, DOM-Based Crawler (DOM-OFF)* [24] is the closest competitor from the literature. It is designed to retrieve *diverse* textual content. It begins by crawling 3 000 pages with BFS, grouping the DOM paths of followed links as in Sec. 3.1. Page benefits are computed immediately, and DOM path groups are stored in a priority queue, ordered by average benefit. After 3 000 pages, [24] *only considers links matching existing DOM path groups*, ordered by priority, ignoring others. In our context, benefit computation requires knowing if links lead to targets, which demands traversal. To enable this baseline, we freely provide the true benefit (number of targets behind a page's links), as if given by an oracle, for the initial 3 000 pages. After 3 000 pages, new DOM path groups can form but are assigned a permanent benefit of zero. Despite its (unfair) first-stage advantage, this baseline acts as an *ablated* version of our crawler, learning *offline*, as opposed to *online* by our RL method.

Finally, the *Focused Crawler* uses a classifier to assess the likelihood that a new hyperlink is a target. It updates the frontier as a *priority queue*, (hopefully) giving priority to targets. It employs a standard *logistic regression* trained periodically on crawled webpages, with no additional HTTP requests. It takes standard input features as in, e.g., [14, 23, 27]: the (approximated) depth of the page where the link was found, the 2-gram vector representation of the hyperlink's URL (as in Sec. 3.3), and the 2-gram vector of the hyperlink's *anchor* (its associated text). We exclude *topic-oriented* features (see Sec. 5 for differences between our problem and traditional focused crawlers). It can also be seen as an *ablated* version of our crawler, lacking HTML link structure information and RL.

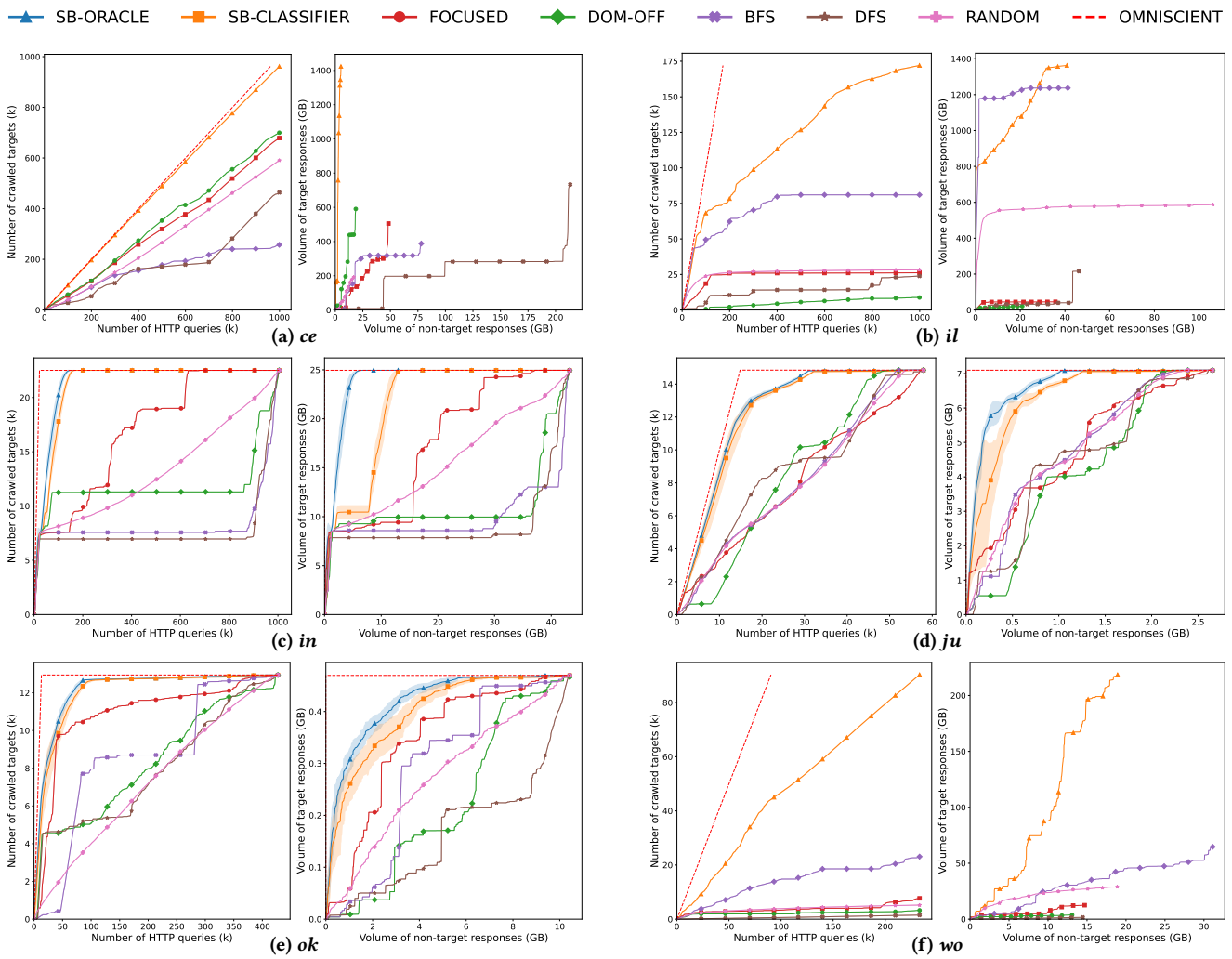


Figure 3: Comparison of different crawler performance on the 6 selected websites presented in Table 1

4.4 Practical Crawling in our Experiments

Evaluating six baselines and our crawler with different hyper-parameters through repeated crawls is impractical due to (a) time constraints (especially with large files and/or slow connections) and (b) crawling ethics. Therefore, for each website, we first run the baselines and our crawler, where each checks if the resource is already in a local database. If found, we use it; otherwise, we fetch it via HTTP GET and save the result (URL, HTTP status, headers, and response body) in the same local database.

For evaluation purposes only, we stop crawling on a given website, as follows: (i) If any crawler finishes before reaching **1M pages**, indicating the entire website was crawled (we have a complete, local replication), all other are stopped and evaluated using local database look-ups. (ii) If each crawler visits 1M pages (possibly not the same pages). (iii) If some crawlers still run beyond the **3 weeks** time limit, set due to practical constraints. Doing so across 18 websites led to the retrieval of **22.2 million distinct webpages**.

We conduct and present the experiments as follows: (i) We carry **hyper-parameter studies** and **evaluate our URL classifier** on websites with *less than 1M pages, fully crawled*. Local availability facilitates multiple runs with different settings. On these websites, we also **gather complete knowledge** for the SB-ORACLE crawler (see Sec. 4.5). (ii) On websites where *all baselines crawled 1M pages*, crawlers are compared **on the subset of the website each crawled**. Note that the target sets and data volumes (target or non-target) may differ; a good crawler selects *target-rich* pages. (iii) For websites where *at least one baseline did not crawl 1M pages within the limit*, crawlers are compared **on the smallest number of pages** visited by any crawler (the first, for those who crawled more). For instance, if three crawlers visit 400k, 200k, and 800k pages, we compare them on their first 200k.

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

4.5 Comparison with Baselines

Figure 3 compares our crawler with the baselines on the selected websites (exhaustive plots appear in [3]). The default settings are: n=2 for n-grams, θ=0.75, α=2$\sqrt{2}$, m=12 (dimension of path vectors), w=15 (in the definition of the hash function), and b=10 (batch size). Two graphs illustrate each crawler's performance. On the left, the *number of crawled targets versus number of HTTP requests* (both GET and HEAD, in thousands); on the right, the *volume of target responses (GB) against the volume of non-target ones*. In both graphs, *a higher curve is better*.

For *fully-crawled websites*, we present two variants of our crawler: SB-CLASSIFIER previously described, and SB-ORACLE, where we replace the URL classifier with a (unrealistic) perfect oracle. This allows assessing the impact of the URL classifier on the crawl performance. Both crawlers results' are *averaged over 15 runs* due to the random link selection and the stochastic nature of our classifier, trained via SGD; standard deviation areas are often, but not always, small. For *partially crawled websites*, we only show a single run of SB-CLASSIFIER, lacking perfect classifier information, and since multiple crawls are unfeasible (Sec. 4.4). All baselines but the random one are deterministic, thus one run suffices.

Table 2 presents performance for each crawler, and all 18 websites. Its left part shows the percentage of queries needed to retrieve 90% of the targets, and its right part the percentage of the volume of non-target pages retrieved, before retrieving 90% of the total target volume. Lower values indicate greater efficiency in reaching the 90% threshold. For websites only partially crawled, we compute these metrics on pages visited by the BFS crawl (as in Table 1). When a crawler never reaches 90%, we show ∞.

We see from Figure 3 and Table 2 (where the best performing crawler is highlighted for each website, excluding SB-ORACLE which does not exist in practice), show that **our crawler, with the actual URL classifier, outperforms all baselines**, with two exceptions. On *cl*, the DFS baseline is slightly better towards the end, and on *is*, FOCUSED reaches the target volume threshold with 10% less non-target volume. On *il*, on volume only, BFS is slightly better for part of the crawl (ours fetches twice more targets, but a lower volume for part of the crawl; over all iterations, our crawler retrieves more target volume anyway). **For websites *as, ce, ed, il, in, ju, nc, wh,* and *wo*, our approach significantly reduces resource usage (time or bandwidth)** to crawl a fixed number of targets, or to crawl as many as possible on a resource budget, demonstrating the practical usefulness of our approach. On other sites (*be*, *cn*, *cl*, *jp*, and *qa*), while some baselines approach our crawler's performance, ours remains superior overall. This shows that **our approach outperforms all considered baselines, on a wide variety of websites** wrt. the size, depth, target distribution, etc. Small sites like *be*, *cl*, *cn*, and *qa* are traversed quickly, allowing less learning, yet our approach adapts effectively.

The offline-trained, DOM-based crawler DOM-OFF, based on [24], performs poorly, particularly on websites like *ed*, *il*, *jp*, *wh*, *wo*. Learning from just 3 000 pages proves insufficient on such large websites. On *cn*, DOM-OFF performs worse than BFS after the initial 3 000 pages, despite there being only three times more pages left. Learning on more pages would probably help, but at the cost of getting closer to a simple BFS crawler. **The focused-crawler**

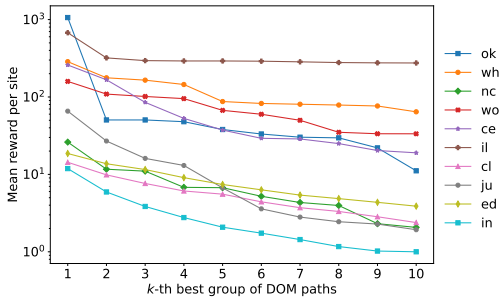


Figure 4: Mean rewards of the top-10 groups of DOM paths for ten selected websites (log y-scale).

FOCUSED is outperformed by ours on all websites, showing it is not adapted to our problem, given their different nature and different purpose. On 10 websites out of 18, it is even beaten by BFS, DFS or RANDOM. Comparing our SB-CLASSIFIER with SB-ORACLE shows that **our classifier is close to the (virtual) perfect oracle**. Further analysis is provided in Sec. 4.7.

4.6 Effectiveness of SB Learning

We first study the reward associated by our algorithm to DOM paths. Figure 4 presents the mean reward of the best 10 path groups (logarithmic y-axis), for a selection of ten websites. We note that **top groups have high rewards**: averaging over all websites we get 258 for the best group, 89 for second, 74 for third, etc., down to 41 for the 10th group. This shows that **our SB agent effectively identifies DOM path groups leading to HTML pages with target links**, i.e., it successfully learns where the reward lies in each website. The (sometimes sharp) difference between first and second group confirms our hypothesis that **links similarly structured in the HTML page where they were found generally lead to similar kind of content**: in our case, **target-rich HTML pages**.

Table 3 shows, for each website, the mean rewards greater than 0 (for HTML pages with links to targets), and their standard deviations. We observe significant disparities in reward distributions, leading to the **impossibility of setting an optimal exploration–exploitation trade-off coefficient** α. Not only are the mean rewards vastly different, but the standard deviations are also large, e.g., for *wo*, more than 20 times its mean. This supports our pragmatic choice of $\alpha = 2\sqrt{2}$, which proves effective in practice.

4.7 URL Classification Quality

The classifier's performance can be summarized by a confusion matrix averaged over 15 runs across 11 fully crawled websites. The classifier correctly identifies 58.04% of HTML URLs, and 32.19% of targets, while misclassifying 0.75% of targets as HTML and 1.37% of HTML as targets. Additionally, 5.34% of "Neither" were misclassified as HTML, and 2.41% as targets. Confusion matrices for all websites are in our report [3]. We see that **classification errors are extremely marginal on "HTML" and "Target" URLs**. Not classifying as "Neither" leads to some classification errors (recall Sec. 3.3), ultimately responsible for the difference between the number of requests made by SB-CLASSIFIER and SB-ORACLE on the fully-crawled websites (Figure 3). However, since the oracle is unfeasible in practice, the performance of SB-CLASSIFIER is satisfactory.

Table 2: Measured performance of different crawlers across all websites

	Percentage of queries for retrieving 90% of the targets																		Percentage of non-target volume before 90% target volume retrieval															
Crawler	*ab*	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*	*ab*	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*
SB-ORACLE	NA	NA	73	NA	71	70	48	NA	13	74	NA	34	51	56	14	49	NA	NA	NA	NA	24	NA	56	25	49	NA	13	58	NA	23	30	48	33	34	NA	NA
SB-CLASS.	**45**	**35**	**84**	**24**	75	**70**	**52**	**14**	**15**	**77**	**39**	**36**	**52**	**59**	**16**	**60**	**20**	**20**	**63**	**21**	**42**	**42**	58	**29**	**50**	56	**27**	84	**40**	**28**	**38**	**50**	**39**	**39**	**17**	**56**
FOCUSED	68	∞	88	36	89	83	87	∞	63	87	42	91	93	85	52	73	∞	∞	∞	∞	85	97	76	75	86	∞	67	**74**	67	72	85	73	50	80	∞	∞
DOM-OFF	96	50	89	35	82	88	96	∞	99	88	∞	74	94	89	76	85	∞	∞	∞	∞	93	64	65	95	93	∞	99	90	∞	72	89	89	74	51	∞	∞
BFS	97	91	89	73	87	80	95	33	99	93	46	81	82	97	67	71	79	92	82	76	67	99	81	51	93	**4**	99	94	54	68	85	98	63	87	92	98
DFS	84	∞	85	75	**71**	85	90	∞	99	88	45	80	94	89	81	75	∞	∞	99	∞	64	97	**45**	82	91	∞	98	85	59	69	96	91	97	80	∞	∞
RANDOM	∞	98	92	45	89	85	95	∞	99	93	∞	83	88	97	85	79	71	∞	72	∞	83	∞	89	83	93	∞	96	98	∞	70	88	98	87	94	∞	∞

Table 3: Mean and standard deviation 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

5 Related Work

Focused crawlers [14, 23] prioritize some pages during the crawl, often based on predefined *topics*. A hyperlink classifier assesses whether a link leads to topic-relevant content. Our approach targets page types instead of topics, relying on website structure rather than textual context or Web graph structure. [25] introduces an adaptive topical focused crawler, also based on MABs, but using a fixed set of always-available estimators. In contrast, our approach involves arbitrarily many actions, necessitating our clustering strategy and the SB model, as not all actions are always usable. [27] brings RL into focused crawling through an MDP [29] with new state and action representations. We differ by using a single-state model (see Sec. 2), and, again by focusing on page types instead of topics. A non-topical focused crawler aimed at content-rich pages is detailed in [24], with an initial training phase followed by intensive exploration. Sec. 4.5 shows that in our setting, this underperforms compared to our Sleeping Bandit approach. [38] presents a focused crawler leveraging MABs for data-rich webpages, while [49] pursues a similar goal by combining MABs with search engines to discover domain-specific datasets. Unlike these works, which prioritize selecting optimal *websites* from thousands, our approach targets specific *pages* within websites, addressing the unresolved challenge of efficiently collecting datasets within each site. Also, [49] is domain-specific and limited to a few thousand pages per domain, whereas our method scales to millions and supports datasets on any topic. Combining these approaches with ours could enhance automated target identification across vast website collections.

Incremental or *revisit* crawlers [16, 18, 39, 43] focus on revisiting already crawled websites looking for new content. Recent methods leverage machine learning: [6, 21, 35] study effective *predictors* (features) of new outlinks in previously visited websites, while [34, 41] use reinforcement learning to quickly identify the new outlinks, for instance. These works are orthogonal to ours, focused on the efficient, static retrieval of targets from a given website.

The literature also covers *parallel* [17] and *distributed* [9, 11, 15] crawling of Web and social networks, or as Web forums [12, 26]; *hidden-* or *deep-Web* crawlers [28] aim for content behind Web forms. These are orthogonal (could be combined) with our work.

Link structures, e.g., *root-to-link* paths in the DOM of each HTML page, have been exploited for Web crawling. [19, 20] use the website structure to cluster webpages, within *Web scrapers* which extract and structure data from HTML pages. [24] directly uses paths in the DOM of each HTML page in order to do focused Web crawling.

While UCB [4] is the state of the art for MABs, simpler approaches like ε-greedy and its variants [44] randomly select actions with probability ε and otherwise choose the highest scoring action. We also considered Thomson Sampling (TS) [46], a probabilistic method based on posterior distribution estimation. We excluded both to ensure our crawler's *stability*; also, TS requires prior distributions, unavailable as we lack prior website knowledge.

6 Conclusion

We addressed the problem of scalable web data acquisition by developing an efficient crawling approach that aims to maximize the retrieval of targets (interesting pages or files) while minimizing computational resource usage in terms of time and bandwidth, and respecting crawling ethics. Our crawler based on RL successfully discovers, and exploits, repeated patterns in the link structure within websites. With no prior knowledge of the website structure, it shows **robust advantages over all baselines**. Our URL classifier is very accurate. Overall, **our crawler is highly effective on numerous, highly heterogeneous websites**, delivering high fractions of each site's (volume of) targets while crawling only a small part.

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, adapting our crawler to *incrementally revisit* sites focusing on new targets is a natural extension, especially for SDs, as new statistics are regularly published.

References

[1] 2024. SDMX Technical Specifications. https://sdmx.org/?page_id=5008.

[2] Ayesh Alshukri, Frans Coenen, and Michele Zito. 2010. Web-Site Boundary Detection. In *Advances in Data Mining. Applications and Theoretical Aspects, 10th Industrial Conference, ICDM*, Vol. 6171. 529–543.

[3] Anonymous. 2024. Efficient Crawling for Scalable Web Data Acquisition (technical report and experiment reproducibility kit). https://anonymous.4open.science/r/efficient_crawling_for_scalable_web_data_acquisition-1D50.

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

[5] 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

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

[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*. ACM, 4798–4802. doi:10.1145/3511808.3557198

[8] Oana Balalau, Ebel Simon, Helena Galhardas, Th éo Galizzi, and Ioana Manolescu. 2024. STaR: Space and Time-aware Statistic Query Answering. In *CIKM 2024 - 33rd ACM International Conference on Information and Knowledge Management* (Boise, Idaho, United States).

[9] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience* 34, 8 (2004), 711–726.

[10] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of 19th International Conference on Computational Statistics*. 177–186.

[11] Matko Bošnjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and s = Sarmento, Luí. 2012. TwitterEcho: a distributed focused crawler to support open research with twitter data. In *Proceedings of the 21st International Conference on World Wide Web*. 1233–1240.

[12] Rui Cai, Jiang-Ming Yang, Wei Lai, Yida Wang, and Lei Zhang. 2008. iRobot: an intelligent crawler for web forums. In *Proceedings of the 17th International Conference on World Wide Web*. 447–456.

[13] 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

[14] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. 1999. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks* 31, 11 (1999), 1623–1640.

[15] Duen Horng Chau, Shashank Pandit, Samuel Wang, and Christos Faloutsos. 2007. Parallel crawling for online social networks. In *Proceedings of the 16th International Conference on World Wide Web*. 1283–1284.

[16] Junghoo Cho and Hector Garcia-Molina. 2000. The Evolution of the Web and Implications for an Incremental Crawler . In *VLDB*, Vol. 2000. 200–209.

[17] Junghoo Cho and Hector Garcia-Molina. 2002. Parallel crawlers. In *Proceedings of the 11th International Conference on World Wide Web*. 124–135.

[18] Junghoo Cho and Hector Garcia-Molina. 2003. Effective page refresh policies for Web crawlers. *ACM Trans. Database Syst.* 28, 4 (Dec. 2003), 390–426. doi:10.1145/958942.958945

[19] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. 2003. Fine-grain web site structure discovery. In *Proc. WIDM*. 15–22.

[20] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. 2005. Clustering Web pages based on their structure. *Data and Knowledge Engineering* 54, 3 (2005), 279–299.

[21] Thi Kim Nhung Dang, Doina Bucur, Berk Atil, Guillaume Pitel, Frank Ruis, Hamidreza Kadkhodaei, and Nelly Litvak. 2023. Look back, look around: A systematic analysis of effective predictors for new outlinks in focused Web crawling. *Knowledge-Based Systems* 260 (2023), 110126. doi:10.1016/j.knosys.2022.110126

[22] Brian D. Davison. 2000. Topical locality in the Web *(SIGIR '00)*. Association for Computing Machinery, New York, NY, USA, 272–279. doi:10.1145/345508.345597

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

[24] Muhammad Faheem and Pierre Senellart. 2015. Adaptive Web Crawling through Structure-Based Link Classification. In *Proc. ICADL*. 39–51.

[25] Georges Gouriten, Silviu Maniu, and Pierre Senellart. 2014. Scalable, generic, and adaptive systems for focused crawling. In *Proceedings of the 25th ACM Conference on Hypertext and Social Media*. 35–45.

[26] Yan Guo, Kui Li, Kai Zhang, and Gang Zhang. 2006. Board forum crawling: a Web crawling method for Web forum. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*. 745–748.

[27] Miyoung Han, Pierre-Henri Wuillemin, and Pierre Senellart. 2018. Focused Crawling Through Reinforcement Learning. In *Web Engineering*. Springer International Publishing, 261–278.

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

[29] Ronald A Howard. 1960. *Dynamic programming and markov processes*. John Wiley.

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

[31] David S Johnson and Michael R Garey. 1979. *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, Chapter A3.1.

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

[33] Robert Kleinberg, Alexandru Niculescu-Mizil, and Yogeshwer Sharma. 2010. Regret bounds for sleeping experts and bandits. *Machine Learning* 80, 2 (2010), 245–272.

[34] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric J Horvitz. 2019. Staying up to Date with Online Content Changes Using Reinforcement Learning for Scheduling. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc.

[35] Andrey Kolobov, Yuval Peres, Eyal Lubetzky, and Eric Horvitz. 2019. Optimal Freshness Crawl Under Politeness Constraints. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Paris, France) *(SIGIR'19)*. Association for Computing Machinery, New York, NY, USA, 495–504. doi:10.1145/3331184.3331241

[36] 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

[37] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[38] 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*. 1039–1048.

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

[40] Mohammed Saeed and Paolo Papotti. [n. d.]. Fact-Checking Statistical Claims with Tables. *IEEE Data Eng. Bull.* 44, 3 ([n. d.]), 27–38.

[41] Peter Schulam and Ion Muslea. [n. d.]. Improving the Exploration/Exploitation Trade-Off in Web Content Discovery.

[42] 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

[43] Kristinn Sigurðsson. 2005. Incremental crawling with Heritrix. (2005).

[44] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, Chapter 1, 2.

[45] 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. https://doi.org/10.1007/978-3-030-49461-2_34

[46] William R. Thompson. 1933. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika* 25 (1933), 285–294.

[47] 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

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

[49] Haoxiang Zhang, Aécio Santos, and Juliana Freire. 2021. DSDD: Domain-Specific Dataset Discovery on the Web. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management* (Queensland, Australia) *(CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 2527–2536. doi:10.1145/3459637.3482427

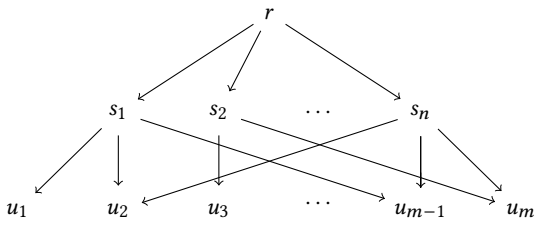


Figure 5: Graphical summarization of the graph G_{sc}

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

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 $\omega(T) \leqslant B$ is NP-complete; hardness holds even when ω is a constant function.*

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

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

We now move to the lower bound. Our crawling problem can be seen as a directed variant of the well-known NP-complete *Steiner Tree* [JG79] problem. NP-hardness of the directed Steiner tree problem is mentioned in the literature (see, e.g., [WW16]), but as it is not formally shown there, we prefer for completeness of the presentation reducing from the set cover problem, a classic NP-hard problem [JG79]. We denote $\mathcal{U} = \{u_1, \ldots, u_m\}$ a set of m elements called the universe. We also define a collection $\mathcal{S} = \{s_1, \ldots, s_n\}$ of n non-empty subsets, each of them containing some elements of $\mathcal{U}$, such that:

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

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

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

We now propose a polynomial-time many-one reduction of the set cover problem to an instance of the graph crawling problem. We create a website graph $G_{\text{sc}} = (V_{\text{sc}}, E_{\text{sc}}, r, \omega, \lambda)$ as follows. We set V_{sc} to be $\{ u_1, \ldots, u_m, r, s_1, \ldots, s_n \}$, including representations for every element of the universe $\mathcal{U}$, every set of the collection $\mathcal{S}$, as well as a distinct root r (by abuse of notation, we do not distinguish between elements of $\mathcal{U}$, $\mathcal{S}$ and the way they are represented in V_{sc}). We define E_{sc} as $\{(r, s_i) \mid i \in \{1, \ldots, n\}\} \cup \{(s_i, u) \mid u \in s_i, i \in \{1, \ldots, n\}\}$. In other words, in G_{sc} from the origin (root) r, we model each element of $\mathcal{S}$ as a vertex, that can be reached following a dedicated (directed) edge. Finally, for each new vertex $s_i \in \mathcal{S}$, we have as many outgoing edges as there are elements of $\mathcal{U}$ in s_i. Finally, we set ω to be the constant function that assigns cost 1 to every vertex, and λ to be some constant function. The result is a graph in the form of a tree of depth 2, depicted in Figure 5. We fix V^* to be $\mathcal{U}$. We state that there exists $C \subseteq \{s_1, \ldots, s_n\}$ such that $|C| \leqslant B$ and $\bigcup_{i \in C} s_i = \mathcal{U}$ if and only if there exists a crawl T_{sc} of G_{sc} containing all elements of V^* and of total cost $\omega(T_{\text{sc}}) \leqslant |\mathcal{U}| + B + 1$.

Let us explain why this reduction is polynomial-time. In set cover, the universe $\mathcal{U}$ can be described by the number m of its elements, with representation size $\Theta(\log m)$. Each set of $\mathcal{S}$ needs to list every element within this set, so a set s_i has representation size $\Theta(\log m \times |s_i|)$. Finally, B has representation size $\Theta(\log B)$. This yields a total input size of $\Theta((\log m)\left(\sum_{i=1}^{n} |s_i| + 1\right) + \log B)$. Note that $\sum_{i=1}^{n} |s_i| \geqslant \max(m, n)$ so this is $\Omega(\max(m, n) \log m + \log B)$. But then, the construction depicted in Figure 5 can clearly be done in time polynomial in m and n (namely, in $O(m \times n)$ in the worst case where every set of the collection contains every element). The reduction is therefore polynomial-time.

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

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

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

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

Table 4: Main characteristics of all websites (see detailed crawling methodology in Sec. 4.4)

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

B Supplementary material for Section 4 (Experimental Results)

This section presents detailed experimental results that could not fit in the paper. Table 4 presents the main characteristics of the 18 websites included in the experiments. Figures 6 to 11 present exhaustive crawler and baselines performance of the 18 websites presented in Table 1. Section B.1 presents a complete hyper-parameter study. Section B.2 presents the detailed performance of the URL classifier.

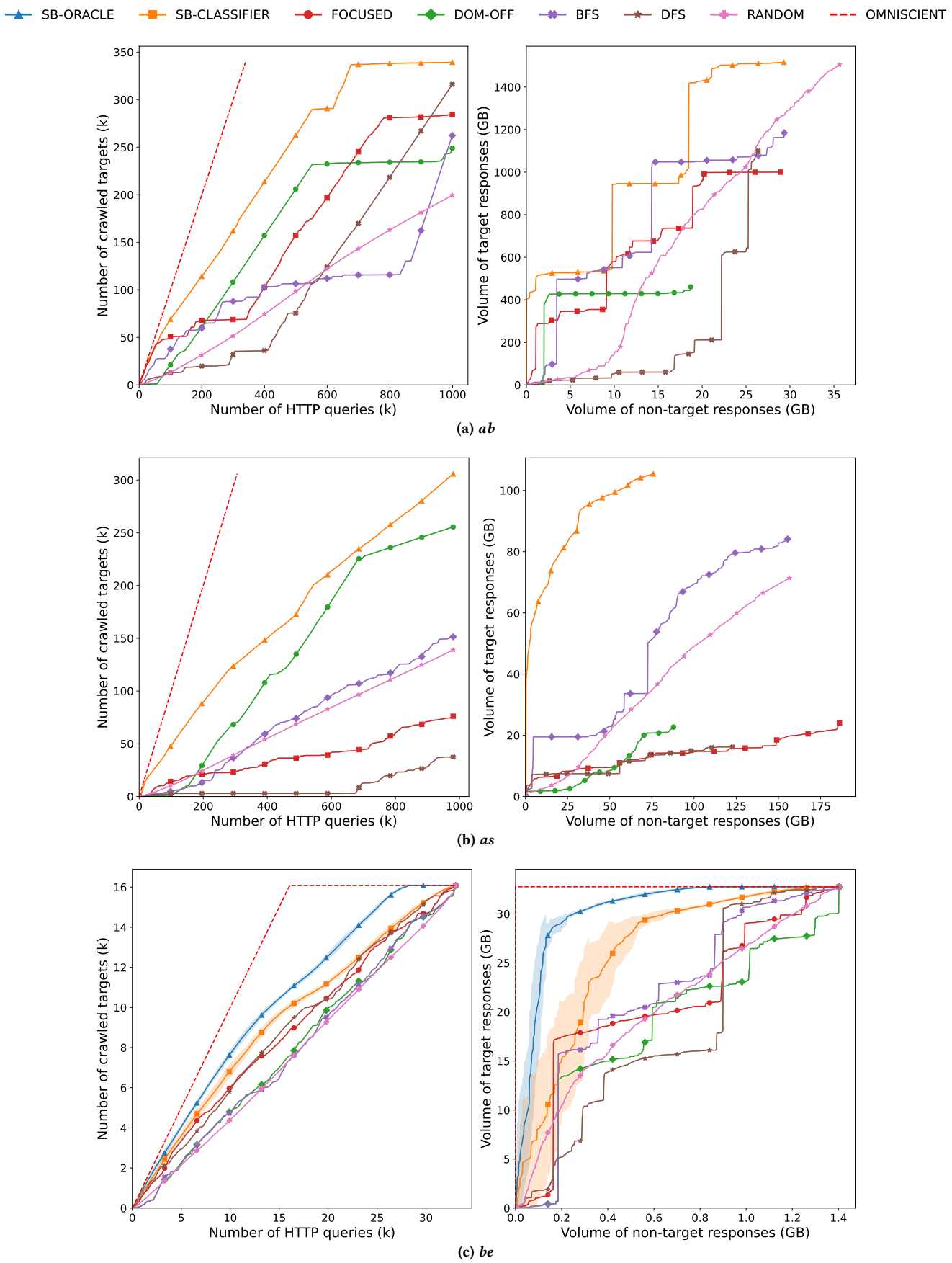
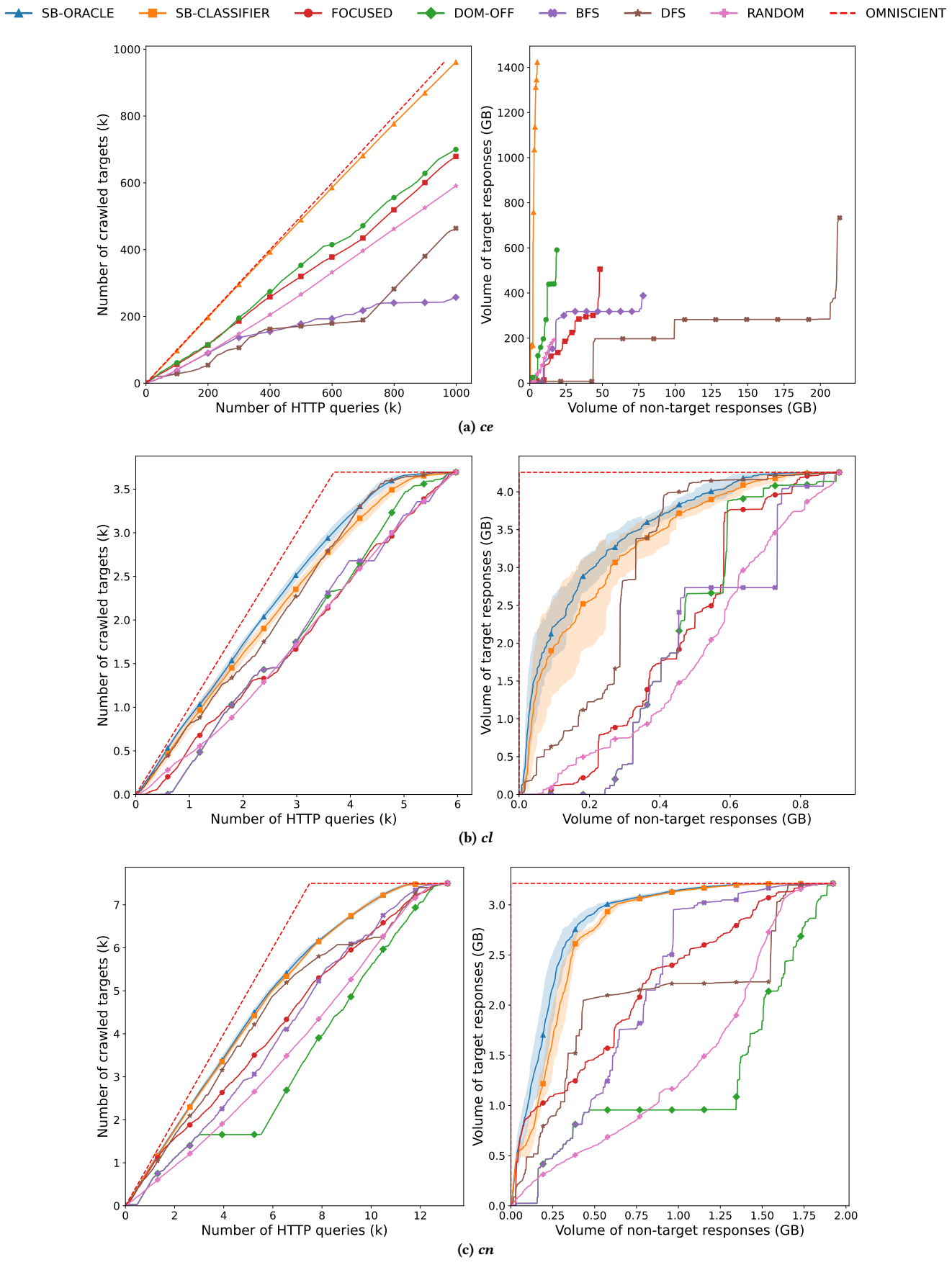


Figure 6: Comparison of different crawler performance for websites *ab*, *as*, and *be*

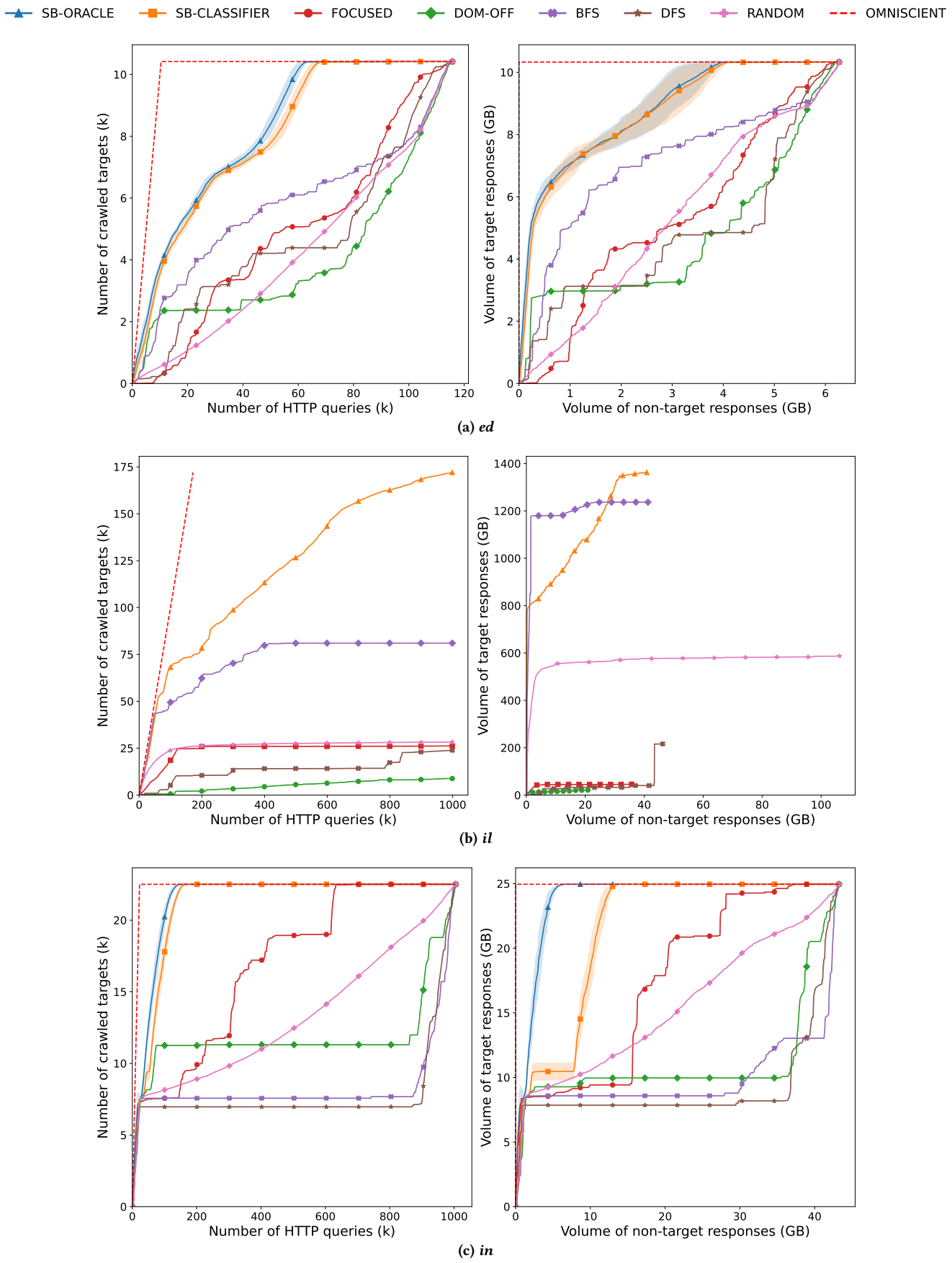


(a) *ce*

(b) *cl*

(c) *cn*

Figure 7: Comparison of different crawler performance for websites *ce*, *cl*, and *cn*



(a) *ed*

(b) *il*

(c) *in*

Figure 8: Comparison of different crawler performance for websites *ed*, *il*, and *in*

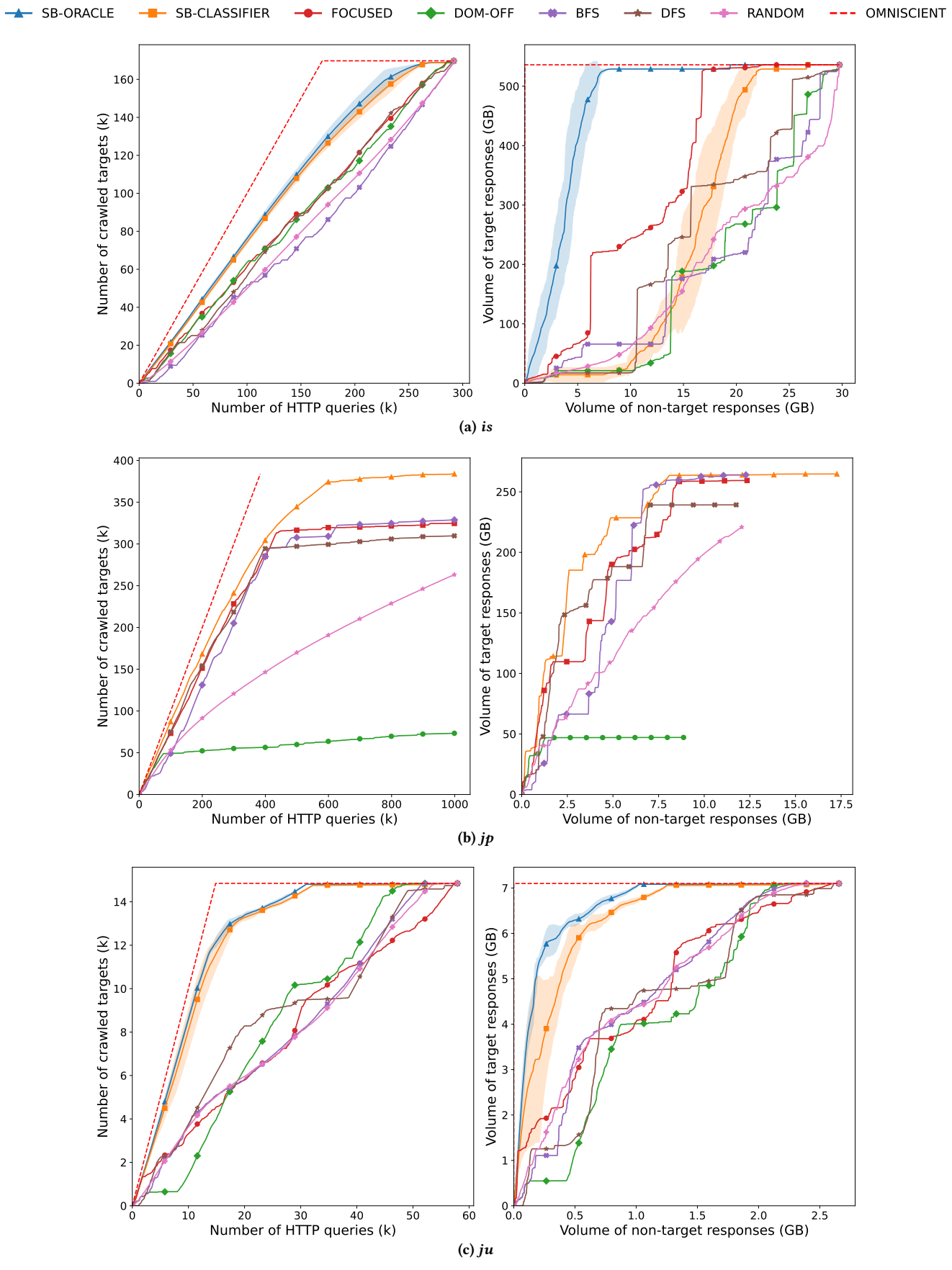
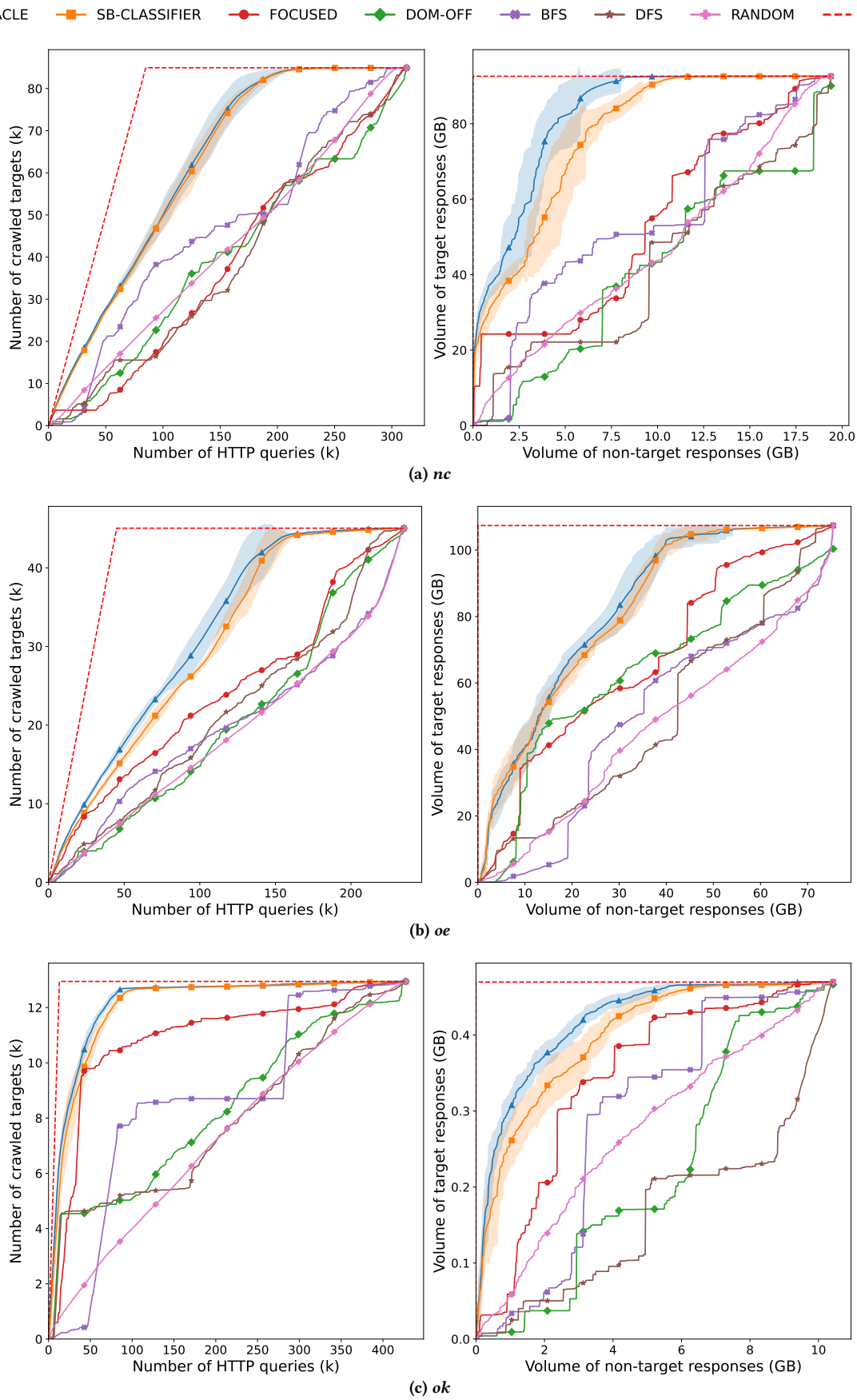


Figure 9: Comparison of different crawler performance for websites *is*, *jp*, and *ju*

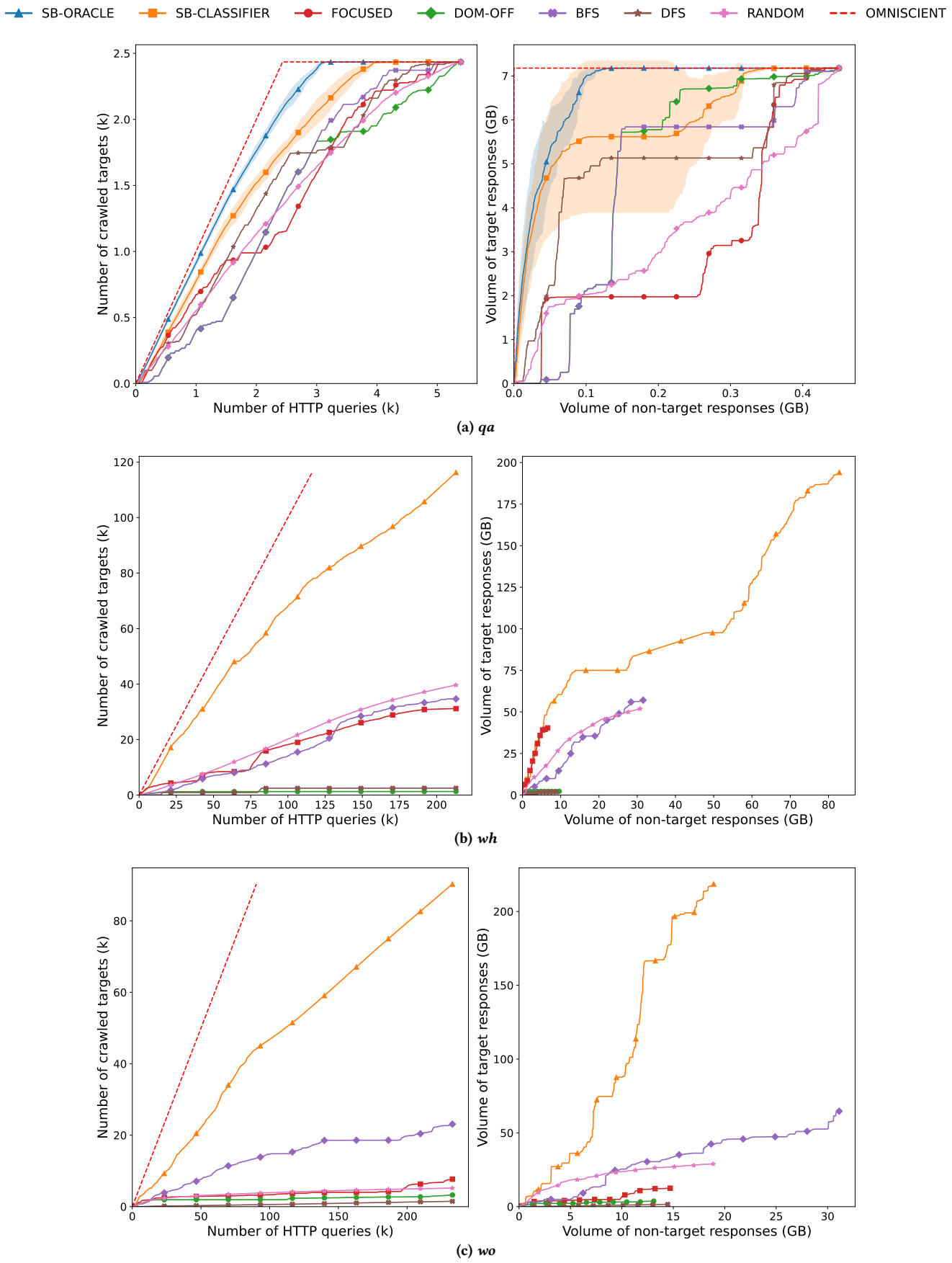


(a) *nc*

(b) *oe*

(c) *ok*

Figure 10: Comparison of different crawler performance for websites *nc*, *oe*, and *ok*



(a) *qa*

(b) *wh*

(c) *wo*

Figure 11: Comparison of different crawler performance for websites *qa*, *wh*, and *wo*

Table 5: Percentage of queries 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 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**

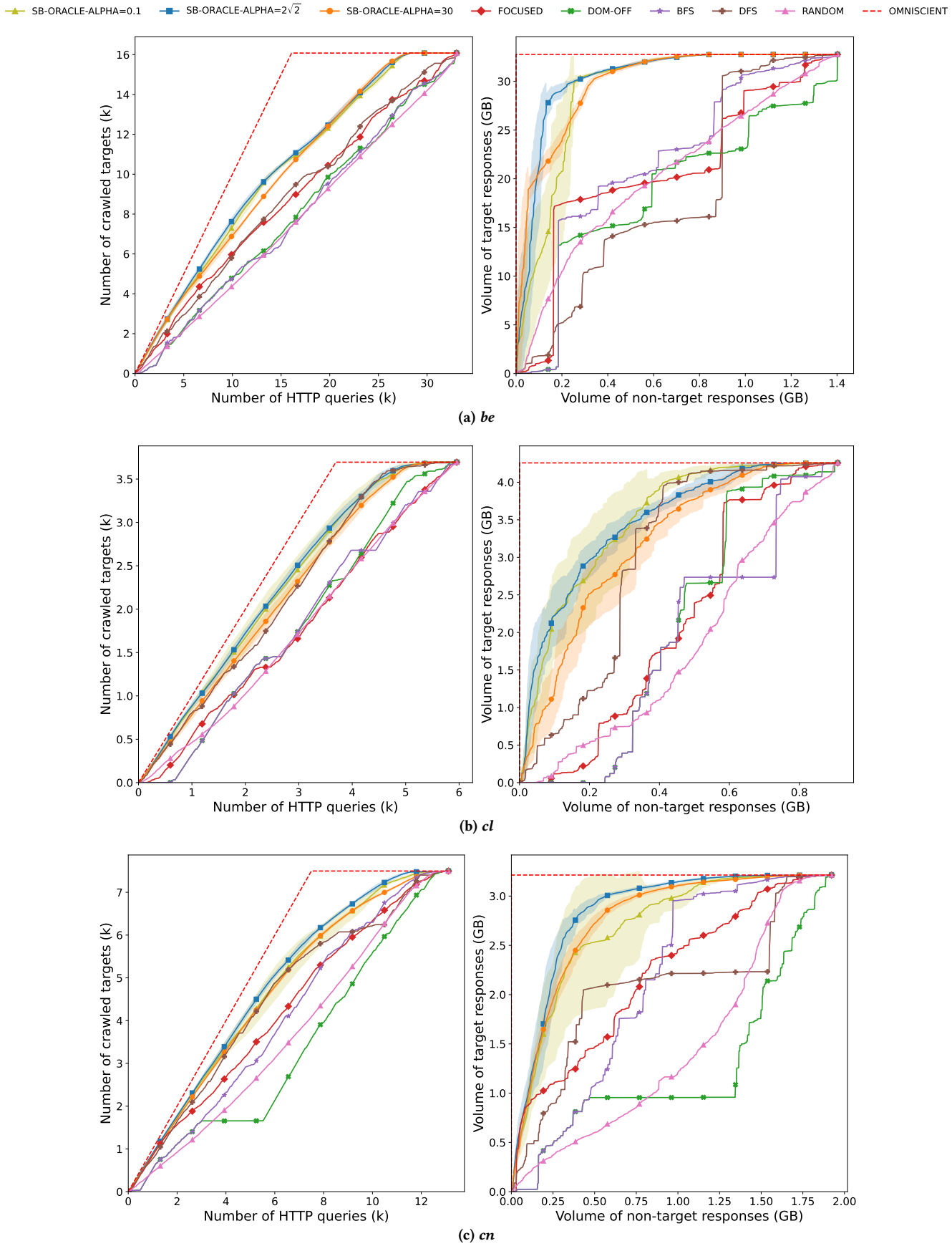
B.1 Hyper-Parameters

Varying four key hyper-parameters, we track the number of queries/volume responses for non-target pages, and requests before retrieving 90% of the total volume of targets. Figures 12 to 23 depict crawler performance regarding hyper-parameter studies on, respectively, exploration–exploitation coefficient α (12 – 15), n in the n-grams used in the DOM path vector representation (16 – 19), and similarity threshold θ (20 – 23); for the 11 fully-crawled websites. Table 5 provides a summary. When varying one hyper-parameter, others keep their default values (see Sec. 4.5).

(1) *Impact of α.* α controls the trade-off between exploration and exploitation (Sec. 3.2). Table 5 (top) shows the effect of varying α in $\{0.1, 2\sqrt{2}, 30\}$: we also try 0.1 and 30 since optimality for $2\sqrt{2}$ is not guaranteed, as our environment does not fit the standard MAB setting. A smaller α generally yields better performance; **best ones obtained with** α=$2\sqrt{2}$. High α values, particularly on websites with moderate or low rewards, lead the crawler to favor *exploration* excessively, neglecting already discovered, useful actions.

(2) *Impact of n in n-grams for DOM path vector representation.* For merging DOM paths (Sec. 3.1), we explored different n-gram representations with $n \in \{1, 2, 3\}$ (n=1 leads to representing a path as a set of HTML tags; n=2 is the default). Table 5 (center) shows that while n=1 works well on some websites, 2 and 3 typically outperform it, confirming our hypothesis that groups of DOM paths provide a better learning basis than sets of HTML tags alone.

(3) *Impact of θ.* The similarity threshold (Sec. 3.1) affects how DOM paths are grouped. We tested low (0.55), high (0.95), and well-performing (0.75) values. Table 5 (bottom) presents this evaluation. A high θ often yields poor outcomes, especially when no similarity value above 95% can be found, as in websites where unique IDs are appended to each tag. For example, it led to an Out Of Memory (OOM) error on *ed*: the learning agent had as many actions as HTML pages in the website. **Good results are achieved with** $\theta \in \{0.55, 0.75\}$**, and** θ=0.75 **works best**. For most websites where θ=0.55 is better, results with θ=0.75 are comparable, but not vice versa; the worst case is *ju*, where θ=0.75 outperforms θ=0.55 by 40% in both number of queries and volume.

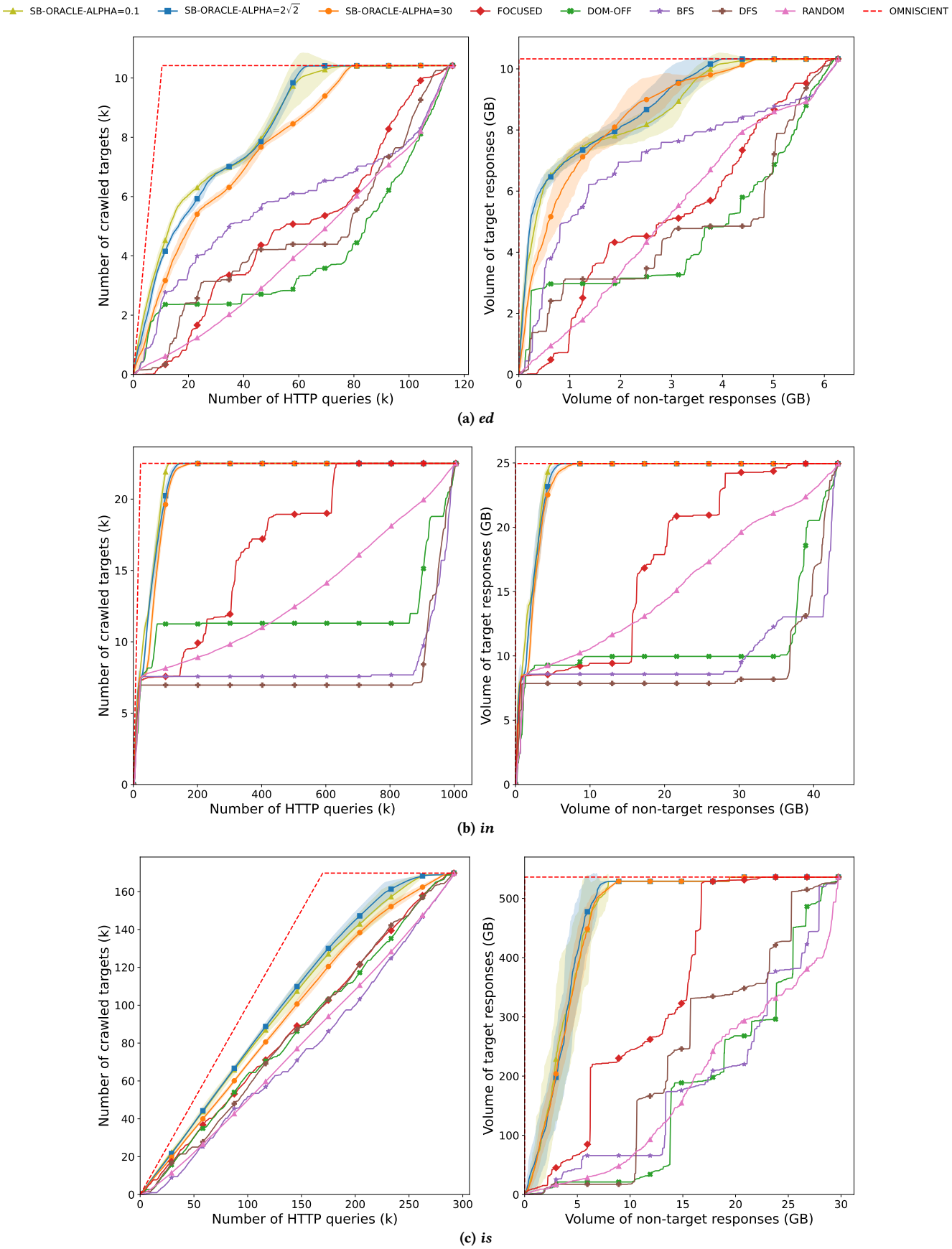


(a) *be*

(b) *cl*

(c) *cn*

Figure 12: Crawler performance for hyper-parameter study on exploration–exploitation coefficent α, for websites *be*, *cl*, and *cn*

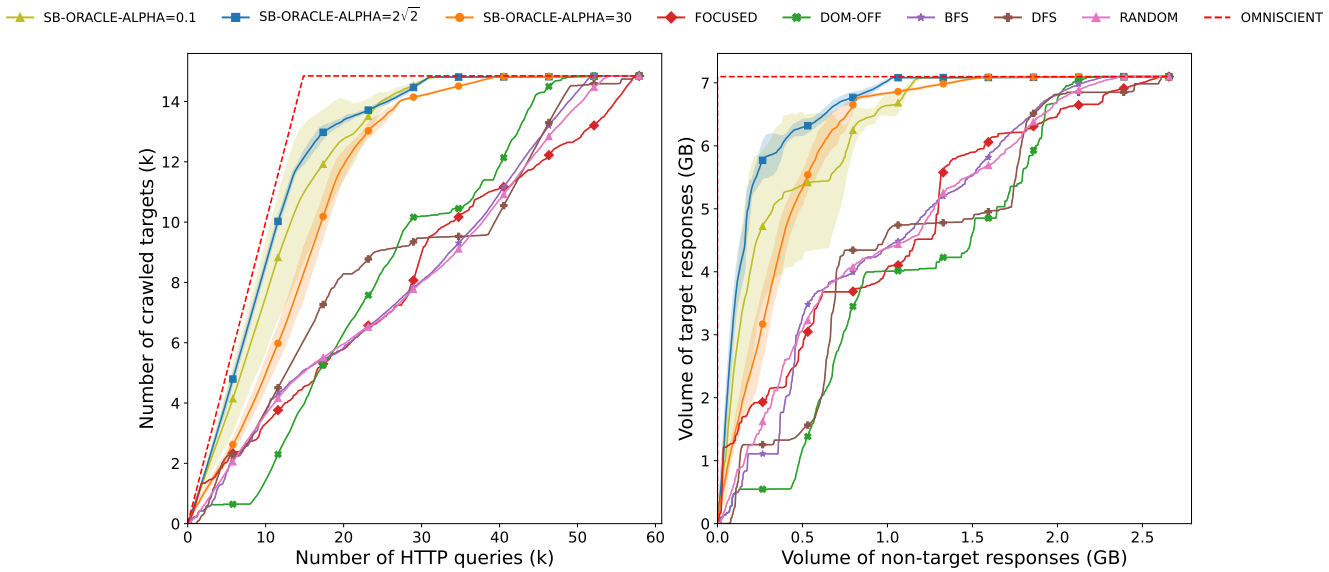


(a) *ed*

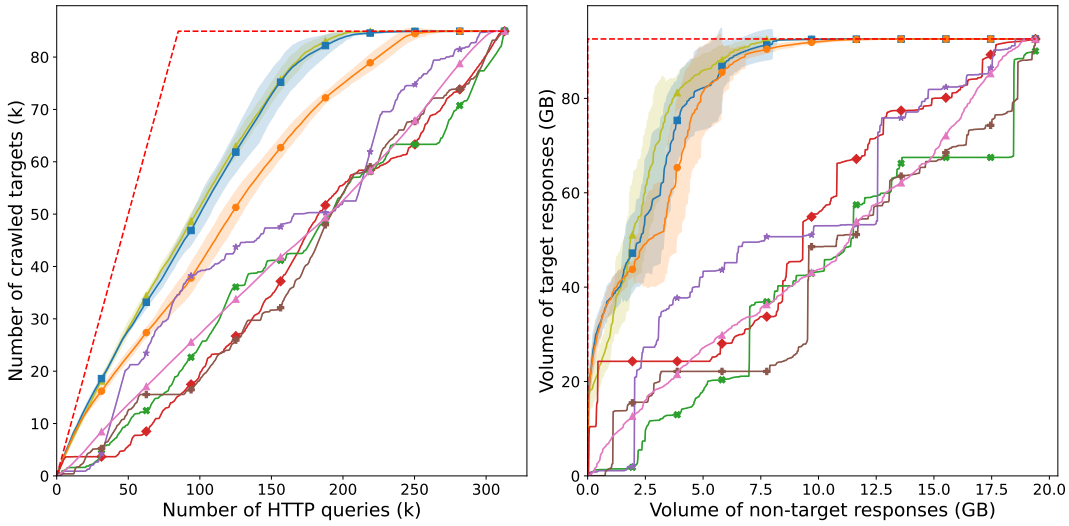
(b) *in*

(c) *is*

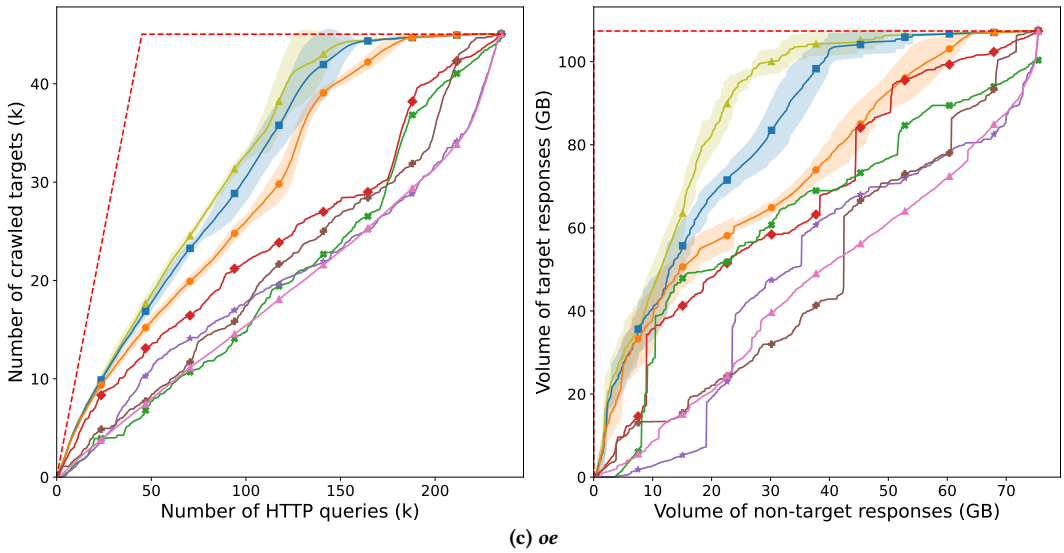
Figure 13: Crawler performance for hyper-parameter study on exploration–exploitation coefficent α, for websites *ed*, *in*, and *is*



(a) *ju*

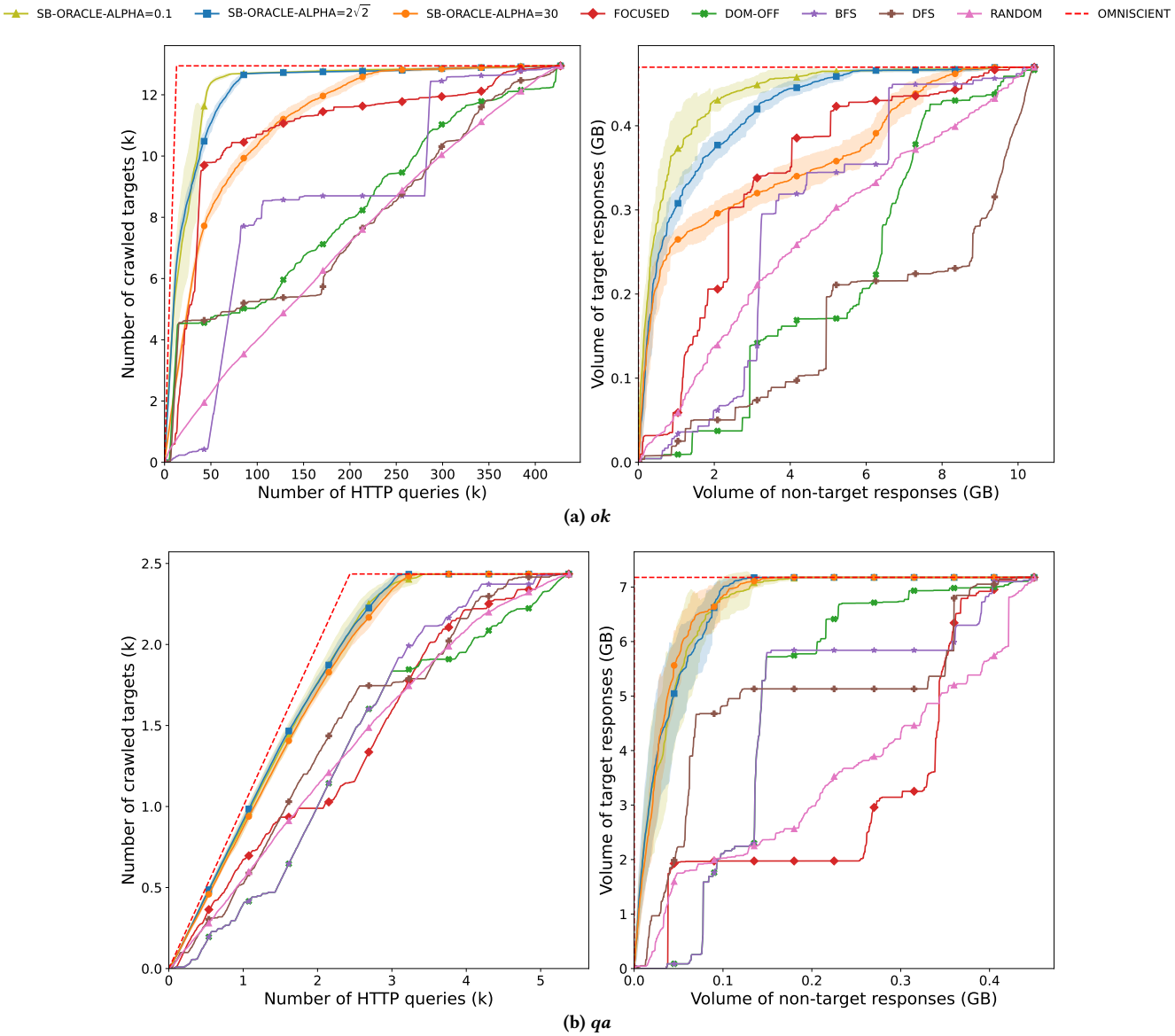


(b) *nc*



(c) *oe*

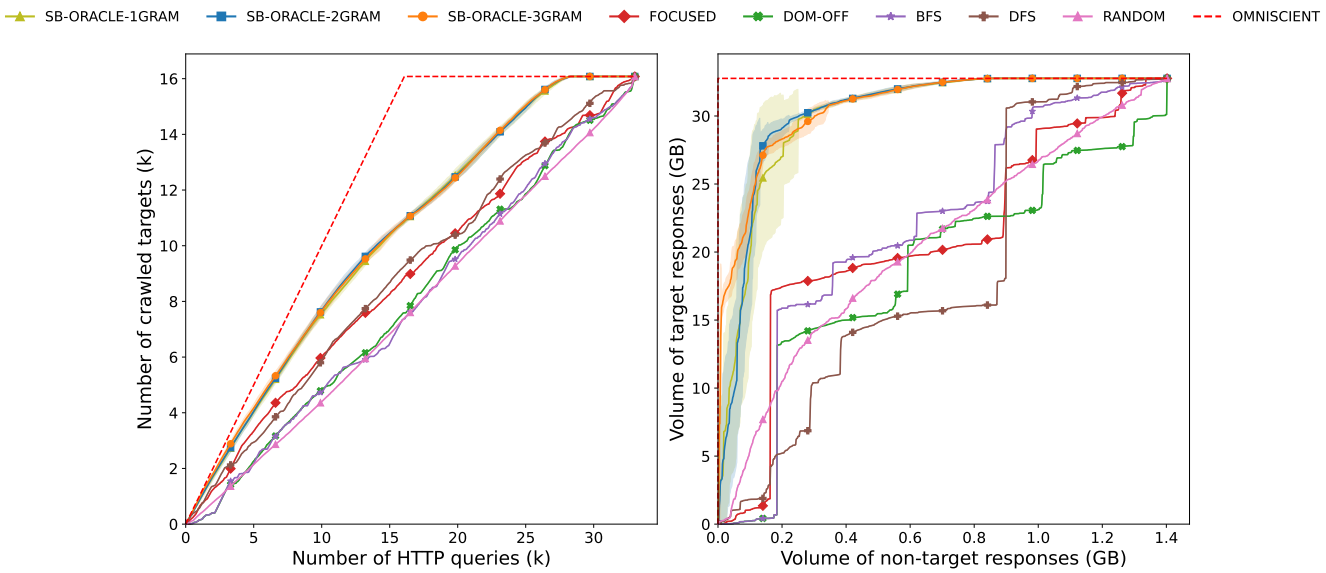
Figure 14: Crawler performance for hyper-parameter study on exploration–exploitation coefficent α, for websites *ju*, *nc*, and *oe*



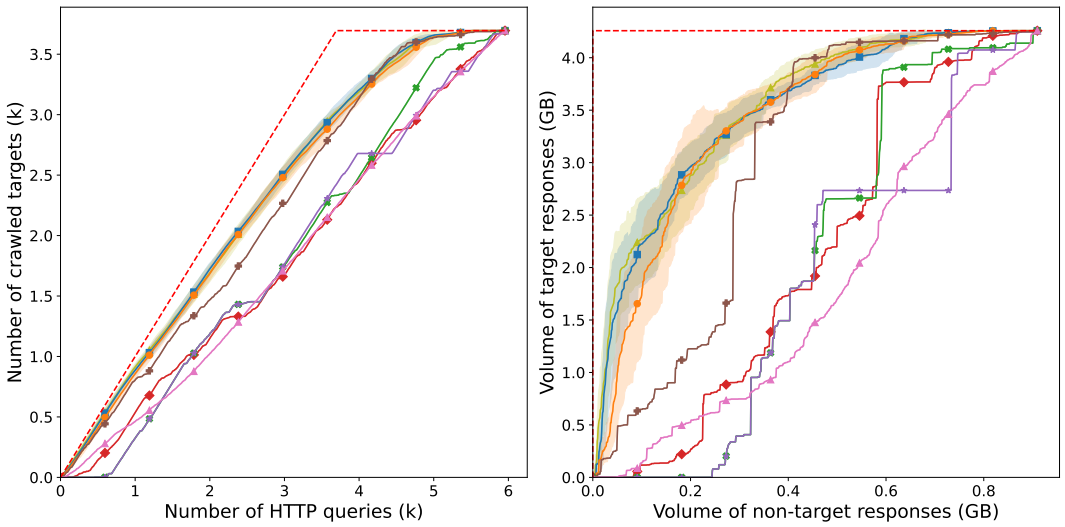
(a) *ok*

(b) *qa*

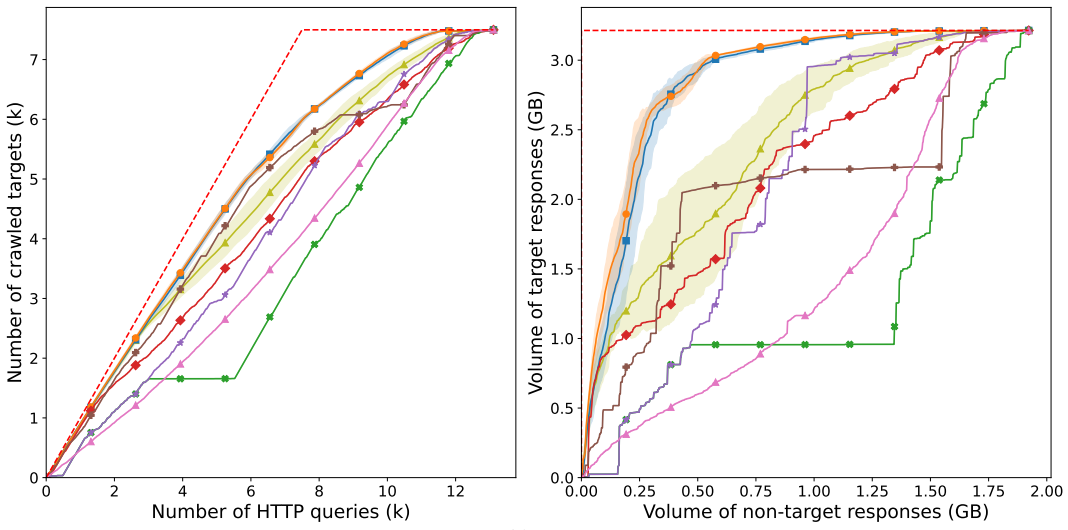
Figure 15: Crawler performance for hyper-parameter study on exploration–exploitation coefficent α, for websites *ok*, and *qa*



(a) *be*

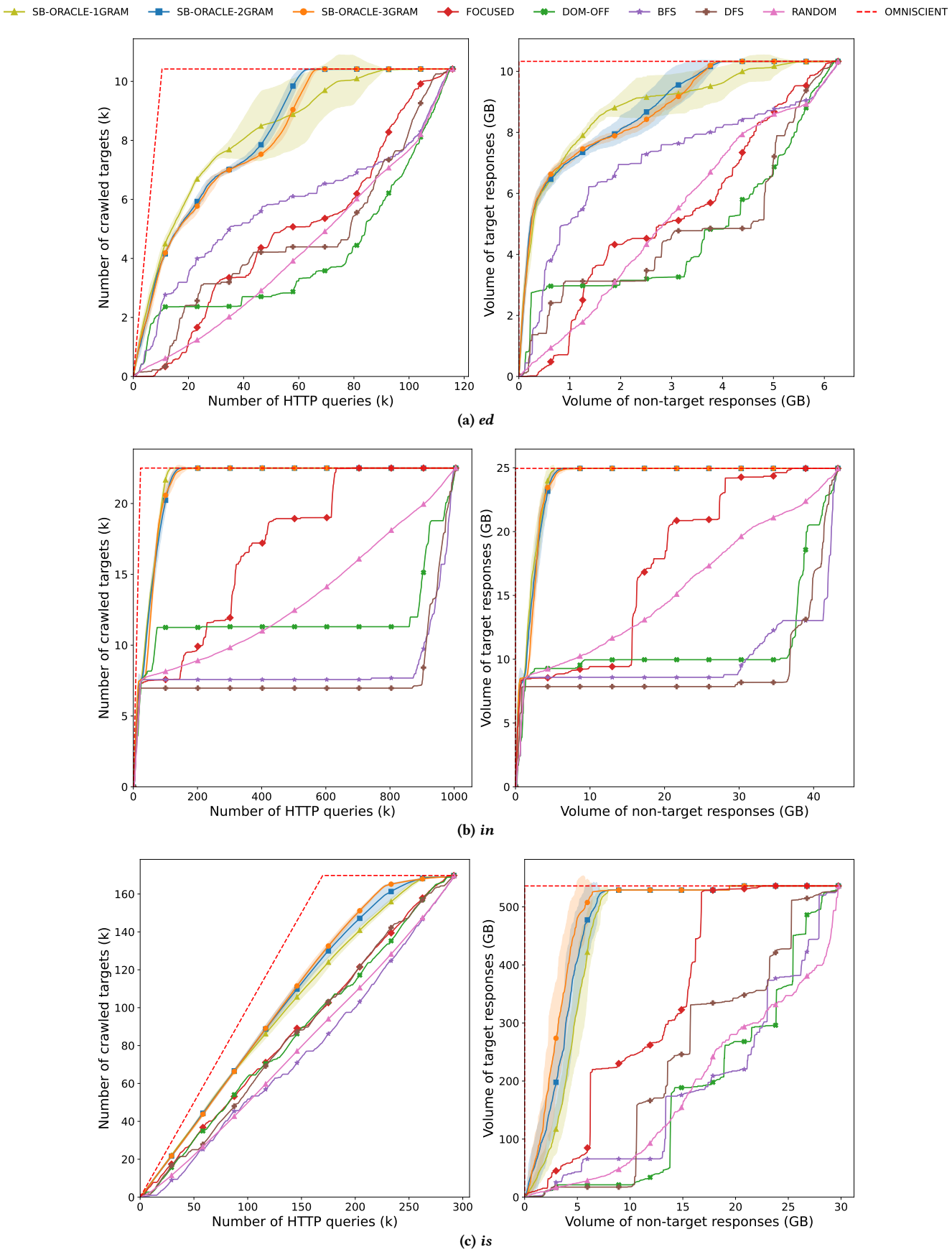


(b) *cl*



(c) *cn*

Figure 16: Crawler performance for impact study of the choice of n in n-grams used in DOM path vector representation, for websites *be*, *cl*, and *cn*



(a) *ed*

(b) *in*

(c) *is*

Figure 17: Crawler performance for impact study of the choice of n in n-grams used in DOM path vector representation, for websites *ed*, *in*, and *is*

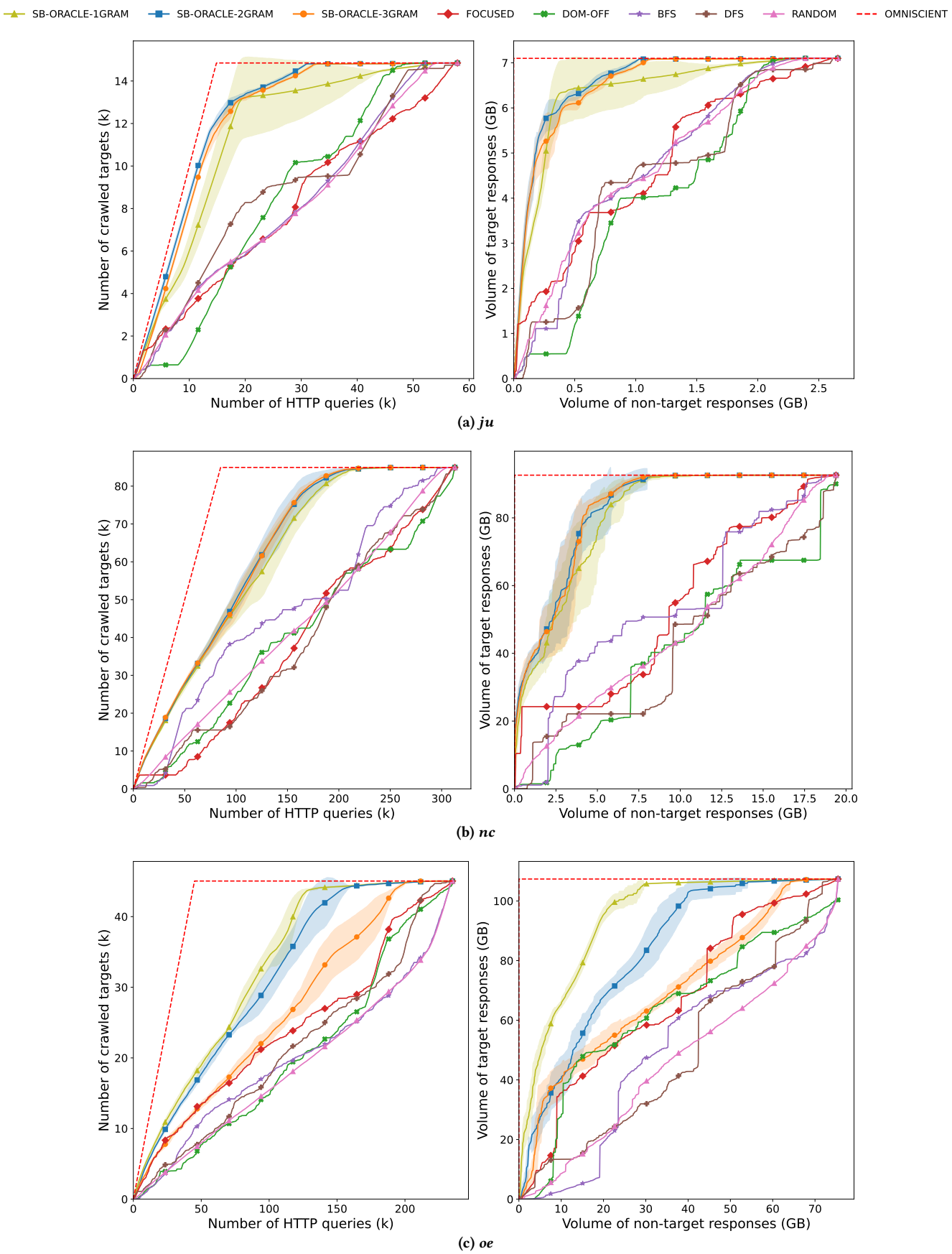
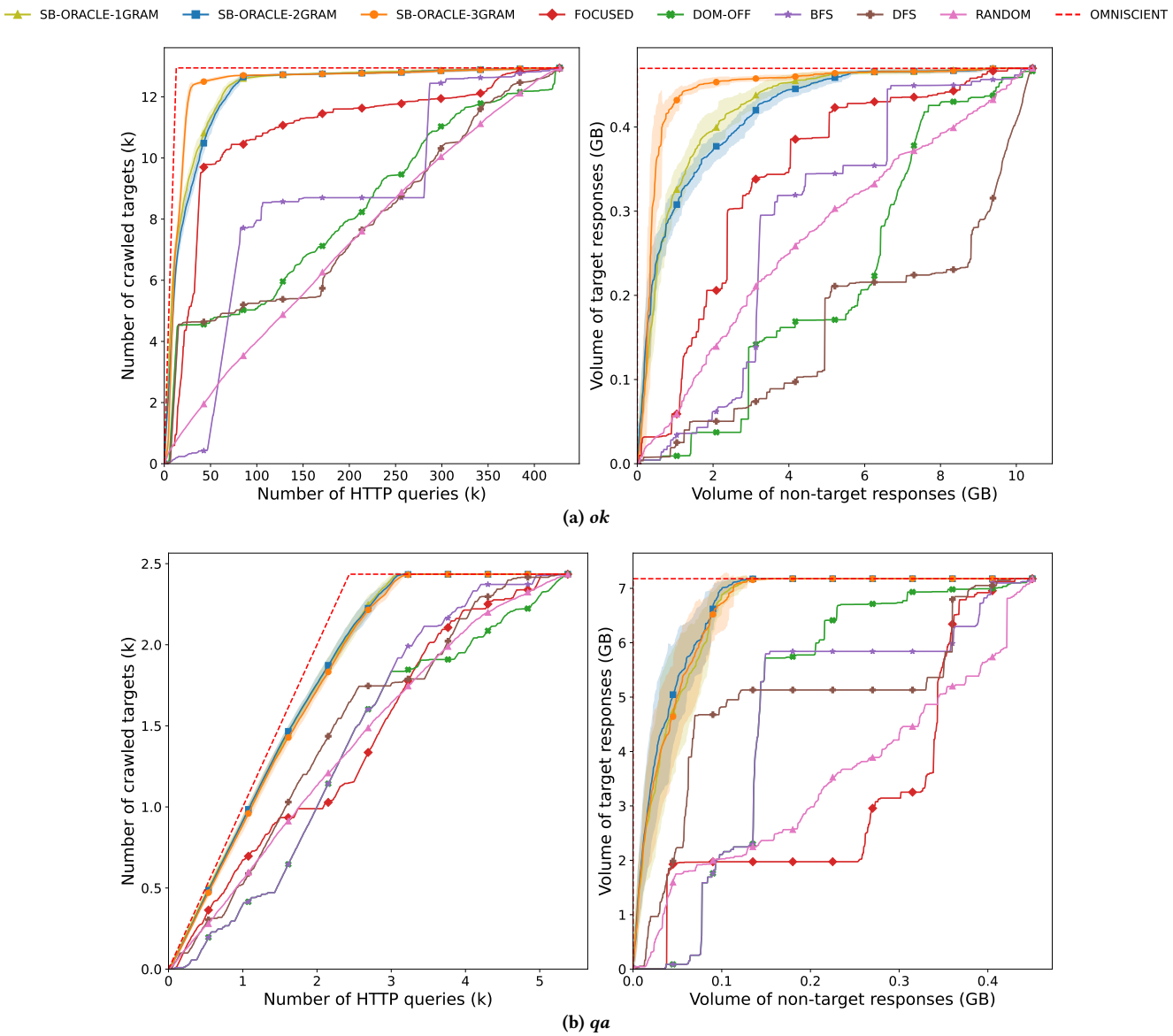


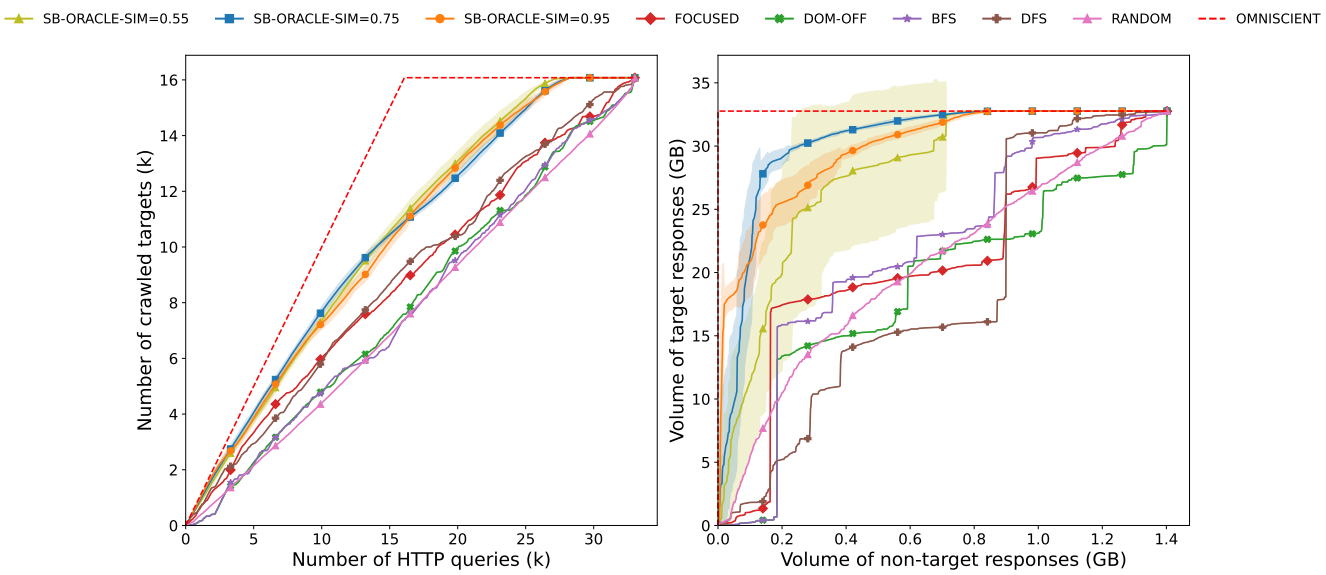
Figure 18: Crawler performance for impact study of the choice of n in n-grams used in DOM path vector representation, for websites *ju*, *nc*, and *oe*



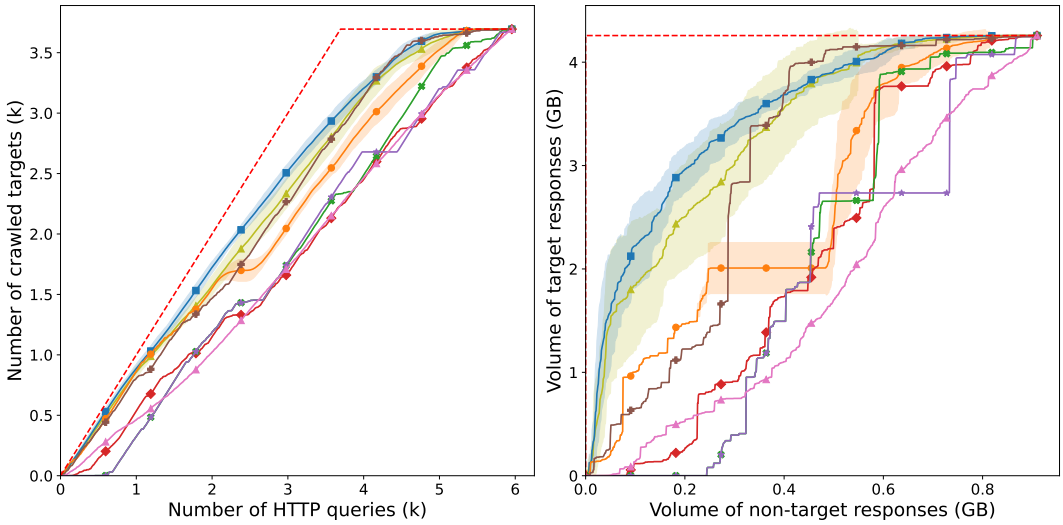
(a) *ok*

(b) *qa*

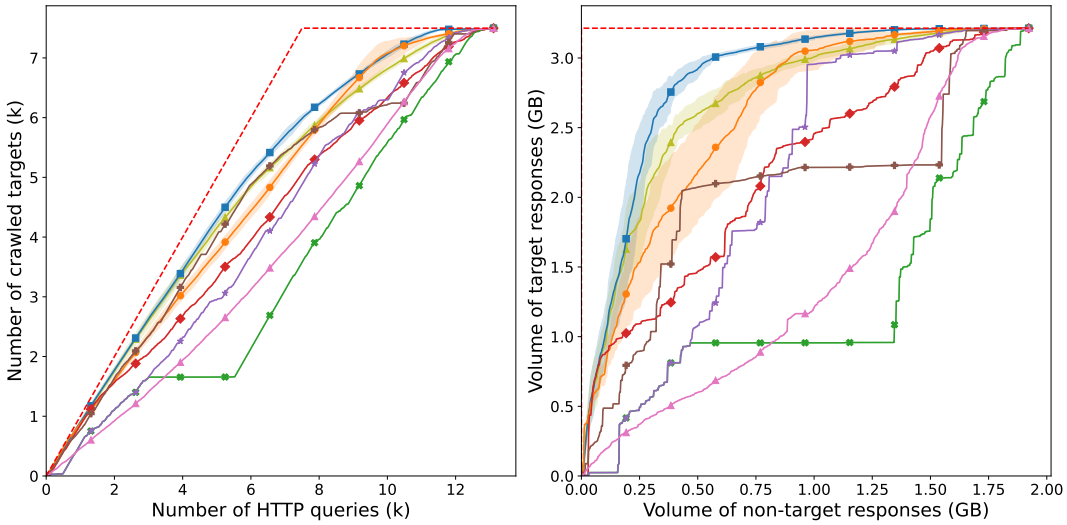
Figure 19: Crawler performance for impact study of the choice of n in n-grams used in DOM path vector representation, for websites *ok*, and *qa*



(a) *be*

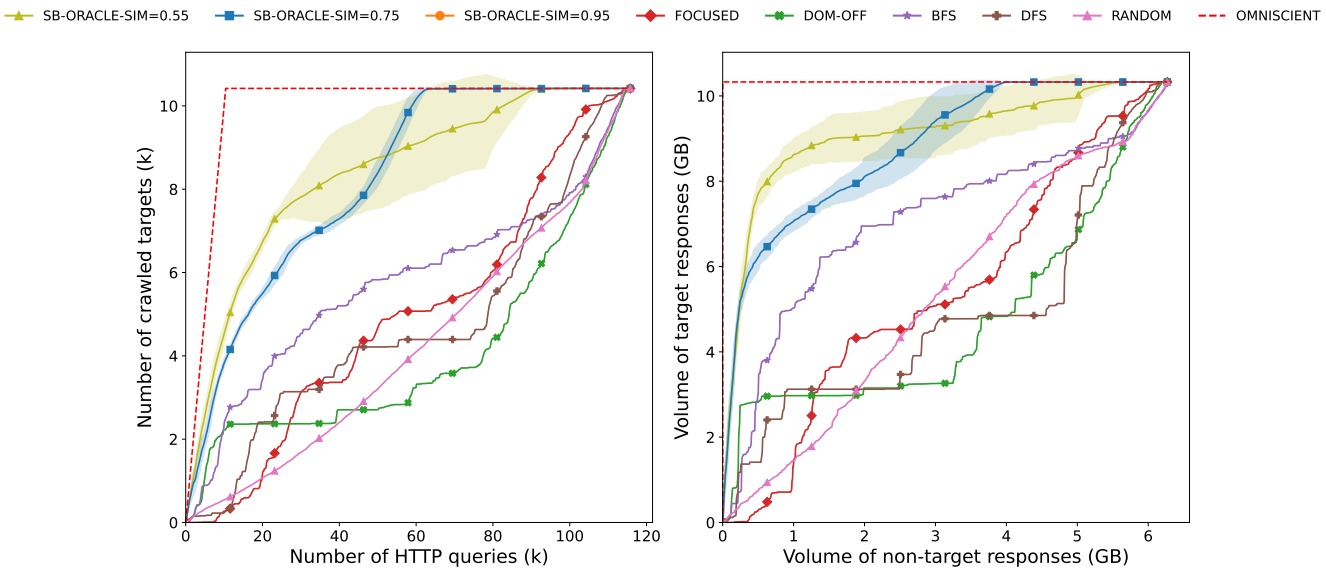


(b) *cl*

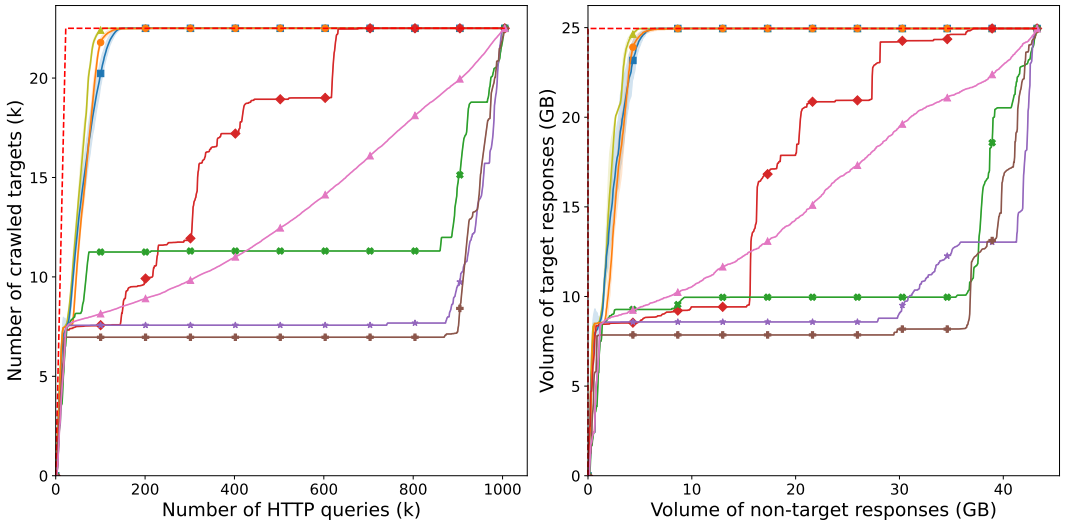


(c) *cn*

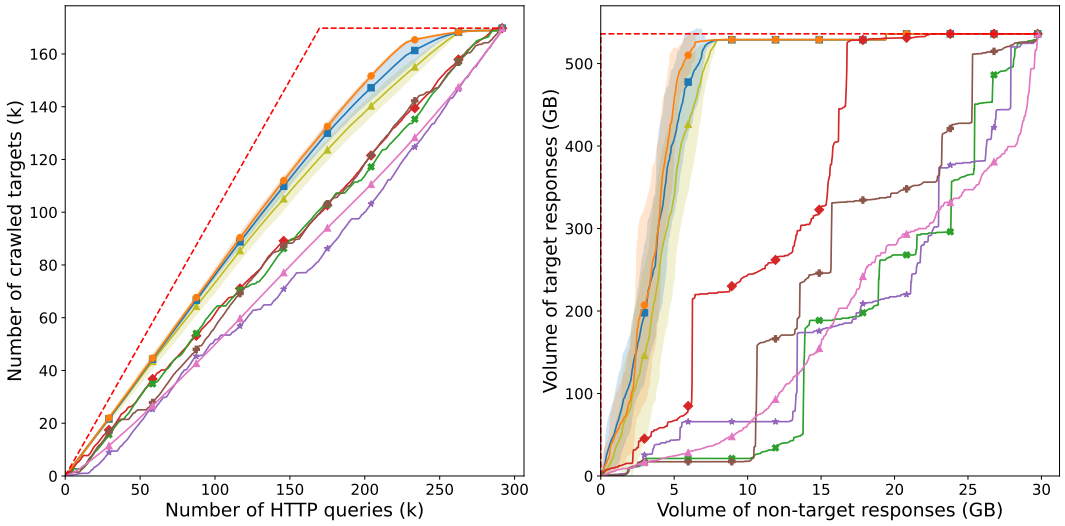
Figure 20: Crawler performance for impact study on similarity threshold θ, for websites *be*, *cl*, and *cn*



(a) *ed*



(b) *in*



(c) *is*

Figure 21: Crawler performance for impact study on similarity threshold θ, for websites *ed*, *in*, and *is*

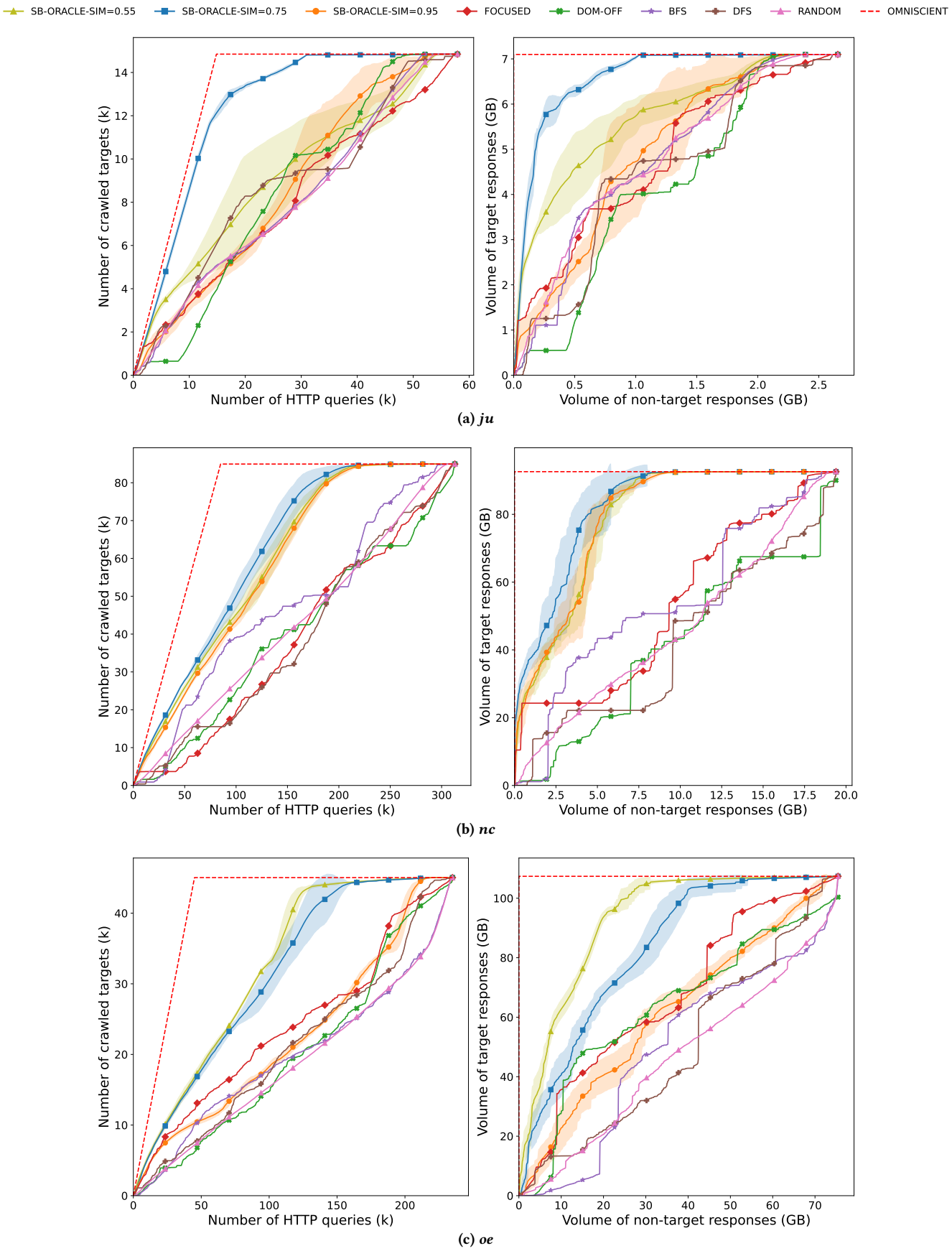
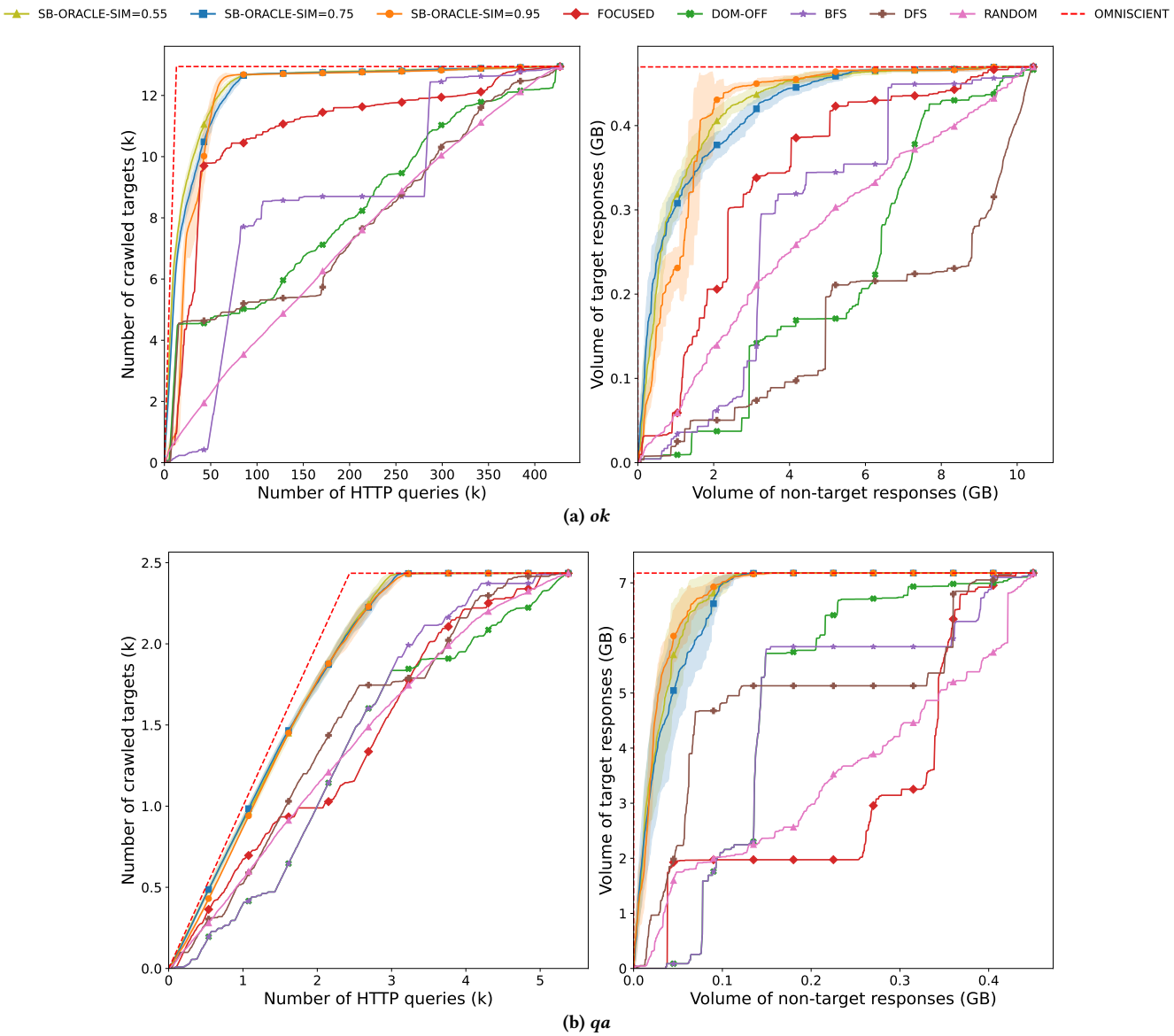


Figure 22: Crawler performance for impact study on similarity threshold θ, for websites *ju*, *nc*, and *oe*



(a) *ok*

(b) *qa*

Figure 23: Crawler performance for impact study on similarity threshold θ, for websites *ok*, and *qa*

B.2 URL Classifier

Tables 6 to 16 present detailed confusion matrices of the URL classifier used in the sleeping bandit algorithm on all websites, averaged for 15 runs (rounded off to nearest unit).

Table 6: Confusion matrix of the URL classifier used in the SB algorithm on website *be* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	11231	272	0
Target	732	15321	0
Neither	2436	2296	0

Table 7: Confusion matrix of the URL classifier used in the SB algorithm on website *cl* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	1654	24	0
Target	154	3538	0
Neither	104	340	0

Table 8: Confusion matrix of the URL classifier used in the SB algorithm on website *cn* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	5272	7	0
Target	64	7423	0
Neither	8	16	0

Table 9: Confusion matrix of the URL classifier used in the SB algorithm on website *ed* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	85920	2610	0
Target	175	10180	0
Neither	8645	1348	0

Table 10: Confusion matrix of the URL classifier used in the SB algorithm on website *in* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	883706	164	0
Target	117	22362	0
Neither	92544	3922	0

Table 11: Confusion matrix of the URL classifier used in the SB algorithm on website *is* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	118805	113	0
Target	120	168762	0
Neither	1185	2292	0

Table 12: Confusion matrix of the URL classifier used in the SB algorithm on website *ju* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	40657	143	0
Target	134	14711	0
Neither	404	1045	0

Table 13: Confusion matrix of the URL classifier used in the SB algorithm on website *nc* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	217531	2193	0
Target	839	82187	0
Neither	3033	1222	0

Table 14: Confusion matrix of the URL classifier used in the SB algorithm on website *oe* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	143412	11071	0
Target	1046	42150	0
Neither	3427	9994	0

Table 15: Confusion matrix of the URL classifier used in the SB algorithm on website *ok* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	391802	3556	0
Target	669	12058	0
Neither	12484	2206	0

Table 16: Confusion matrix of the URL classifier used in the SB algorithm on website *qa* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	1311	31	0
Target	77	2356	0
Neither	1320	247	0

C Supplementary material for Section 5 (Related Work)

We discuss in detail different families of crawler research that is mostly orthogonal to the problem of Web data acquisition.

(1) *Distributed* or *parallel* Web crawlers. Their principle is not to use a single crawler for the crawling task, but to use several simultaneously, so as to minimize the time it takes to crawl a fixed number of pages. The main challenges of this type of crawler lies in resource management, and particularly network resources. [CGM02] first introduces parallel crawlers, describing a general architecture. [BCSV04] presents "UbiCrawler", a decentralized and distributed crawler exploiting consistent hashing to partition the domains to be crawled between several servers. [CPWF07, BOM+12] address the use of distributed crawlers for social networks retrieval. This family of crawlers pursue different objectives as ours: theirs is to optimize resources as much as possible in order to crawl a predetermined, fixed set of pages while ours focuses on minimizing this set of pages to be crawled. It is therefore possible to use a distributed crawler as an overlay to ours, in order to optimize crawling time and network infrastructure usage, in addition to minimizing the number of requests sent to the server and the total volume of data exchanged, both of which being invariant to the choice of the said distributed crawler.

(2) Web crawlers concentrating on specific types of websites. Such crawlers address the crawl of forums, blogs, *Content Management Websites* (CMS), etc. They are built to take advantage of the specific structuring of these types of websites, which often do not vary from one to another. For instance, [GLZZ06, CYL+08] present crawlers that are taking advantage of the specific content and structure of forums. The crawlers are therefore less general than the one we present. We also take advantage of the structure of the websites, but without making any prior assumption about it, just reasoning over similarity between already visited webpages of each website (more details can be found in Sec. 2). Moreover, the websites that such crawlers are targeting, especially forums and blogs, are not the most natural candidates for extracting large amounts of targets; let alone focusing on the acquisition of *trustworthy* targets.

(3) *Hidden-* or *deep*-Web crawlers. They postulate that the Web might not be accessible just following hyperlinks on HTML pages, but also behind interfaces where an interaction with the user is required. Most of the time, these interactions are forms to be filled and sent, query interface, search interface, etc. [HRR19] presenting most state-of-the-art deep-Web crawlers, as well as a framework allowing comparison of such crawlers regarding a wide range of aspects. Our approach does not cover this type of crawling, and therefore constitutes a natural extension to our work (as discussed in Sec. 6). The choice of putting aside this problem lies in the main application of our crawler, that is acquiring specific types of targets. As we focus on official sites providing public statistical data, most encountered forms are filters in the form of portals, build to target specific data based on subject, location, format, etc. In a context where we want to massively retrieve targets, we have no interest in using these filtering forms. In addition, we observe that on sites that are not specialized in the provision of statistical data (such as the sites of French ministries, more detail in Sec. 4.1), these targets are generally not accessible through a portal, but rather by navigating through the links.

(4) *Incremental* or *revisit policy-based* crawlers. They postulate that the Web is not a static collection of pages, but a dynamic one: therefore the pages of a given website are likely to evolve over time. Thus, there is a need for revisiting some of the pages. The challenge here is to be able to minimize the number of revisited pages while maximizing the retrieval of updated content. [CGM00] presents the most important challenges regarding incremental crawling and how they should be influenced by the evolution of the Web. [Sig05] presents an incremental version of the *Heritrix* project [MSR+04], *Internet Archive*'s[3] open-source, extensible, web-scale, archival-quality Web crawler. Ours is not built to handle such an incremental crawling. Instead, it retrieves targets from an unknown website, with an unknown structure, that is to be discovered in an online fashion. This is called *snapshot* crawling, as opposed to incremental. We are still planning on exploiting the result of this snapshot crawling phase, and especially the parts of the website that are the most fruitful, so that we can later on do some incremental crawling. This is particularly important since the retrieved data are provided at a given time t, and, if trustworthy, are only relevant at that time. Other, more recent works leverage modern machine learning techniques to the end of incremental crawling. [DBA+23, APT22, KPLH19b] try to identify the most effective *predictors* of new outlinks in already crawled pages (i.e., new links that were added to the website). The rely on different families of features: *static page* (SP) features (e.g., page content-size, number of internal or external links, etc.), *dynamic page* (DP) features (mostly evolution of SP features through different re-crawls), *static network* features (SN) (*TrustRank* [GGMP04], features extracted from the graph structure of inlinks, etc.), and *dynamic network* (DN) features (mostly evolution of SN features through different re-crawls). [SM, KPLH19a] focus on leveraging reinforcement learning to effectively choose the pages to re-visit. [SM] shows that Thompson Sampling experimentally outperform Multi-Armed Bandits and the *Pham Crawler* [PSF18] on this task, while [KPLH19a] formalize a new *optimization objective* for this setting, along with a leanrning algorithm called LambdaCrawl that founds an optimal policy for this objective, shown efficient in practice.

References for the Appendix

[APT22] Konstantin Avrachenkov, Kishor Patil, and Gugan Thoppe. Online algorithms for estimating change rates of web pages. *Performance Evaluation*, 153:102261, 2022.
[BCSV04] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience*, 34(8):711–726, 2004.
[BOM+12] Matko Bošnjak, Eduardo Oliveira, José Martins, Eduarda Mendes Rodrigues, and s = Sarmento, Luí. Twitterecho: a distributed focused crawler to support open research with twitter data. In *Proceedings of the 21st International Conference on World Wide Web*, page 1233–1240, 2012.
[CGM00] Junghoo Cho and Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler . In *VLDB*, volume 2000, pages 200–209, 2000.

[3] https://archive.org/

[CGM02] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, page 124–135, 2002.

[CPWF07] Duen Horng Chau, Shashank Pandit, Samuel Wang, and Christos Faloutsos. Parallel crawling for online social networks. In *Proceedings of the 16th International Conference on World Wide Web*, page 1283–1284, 2007.

[CYL+08] Rui Cai, Jiang-Ming Yang, Wei Lai, Yida Wang, and Lei Zhang. iRobot: an intelligent crawler for web forums. In *Proceedings of the 17th International Conference on World Wide Web*, page 447–456, 2008.

[DBA+23] Thi Kim Nhung Dang, Doina Bucur, Berk Atil, Guillaume Pitel, Frank Ruis, Hamidreza Kadkhodaei, and Nelly Litvak. Look back, look around: A systematic analysis of effective predictors for new outlinks in focused Web crawling. *Knowledge-Based Systems*, 260:110126, 2023.

[GGMP04] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating Web Spam with Trustrank. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, page 576–587. VLDB Endowment, 2004.

[GLZZ06] Yan Guo, Kui Li, Kai Zhang, and Gang Zhang. Board forum crawling: a Web crawling method for Web forum. In *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI'06)*, pages 745–748, 2006.

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

[JG79] David S Johnson and Michael R Garey. *Computers and intractability: A guide to the theory of NP-completeness*, chapter A2.1. WH Freeman, 1979.

[KPLH19a] Andrey Kolobov, Yuval Peres, Cheng Lu, and Eric J Horvitz. Staying up to Date with Online Content Changes Using Reinforcement Learning for Scheduling. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[KPLH19b] Andrey Kolobov, Yuval Peres, Eyal Lubetzky, and Eric Horvitz. Optimal freshness crawl under politeness constraints. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 495–504, New York, NY, USA, 2019. Association for Computing Machinery.

[MSR+04] Gordon Mohr, Michael Stack, Igor Rnitovic, Dan Avery, and Michele Kimpton. Introduction to Heritrix. In *4th International Web Archiving Workshop*, pages 109–115, 2004.

[PSF18] Kien Pham, Aécio Santos, and Juliana Freire. Learning to Discover Domain-Specific Web Content. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 432–440, New York, NY, USA, 2018. Association for Computing Machinery.

[Sig05] Kristinn Sigurðsson. Incremental crawling with Heritrix. 2005.

[SM] Peter Schulam and Ion Muslea. Improving the exploration/exploitation trade-off in web content discovery.

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