Efficient Crawling for Scalable Web Data Acquisition

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

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

Pierre Senellart
DI ENS, ENS, CNRS, PSL University, Inria & IUF
Paris, France
pierre@senellart.com

***Abstract*—Open Data has enabled many applications, e.g., in science, in daily-needs apps such as finding nearby restaurants or bikes to rent, and data processing tasks enabled or enhanced by Open Knowledge Bases. Journalistic fact-checking, as well as social or economic research, require analyzing high-quality datasets which may be very hard to find, either manually, or via search engines or existing crawlers. To improve Open Data accessibility, we present a *focused Web crawling algorithm* that retrieves as many *targets*, i.e., resources of certain types, as possible, from a given website, in an efficient and scalable way, by crawling (much) less than the full website. We show that optimally solving this problem is intractable, and propose an approach based on reinforcement learning, namely using sleeping bandits. Our algorithm efficiently learns *which hyperlinks lead to targets*, based on the path leading to the links in their enclosing webpages. Our experiments on websites with millions of webpages show that our crawling is highly efficient, delivering high fractions of a site's targets while crawling only a small part.**

I. Introduction

Open Data designates datasets whose use is open to all, without access fees or legal restrictions. The history of Open Data is closely connected with the Web as a platform for content sharing. In particular, HTML pages containing tables have been leveraged to extract large-scale, open knowledge bases, e.g. [1], [2], [3], [4]. 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. Such datasets 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**. Different from relations in which each line is about an entity, e.g., album, whose columns describe 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 [5], etc; and sometimes embedded in PDF documents, or serialized in JSON. Some organizations, e.g., Eurostat, publish hundreds of thousands of SDs, updated early; others, e.g., a watchdog NGO focused on equitable justice, 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 and detect trends; newsrooms leverage SDs for their analysis and to check the accuracy of public statements. Research works seeking to automatize the recognition and checking of claims based on SDs include, e.g., [6], [7], [8], [9], [10]. In particular, we collaborate [9] with journalists from France's national RadioFrance group: they need to use SDs from national and EU sources for fact-checking and to put national analysis into perspective.

To leverage Open SDs, one first needs to *acquire* them. 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 **a tiny fraction only** of existing SDs (Sec. IV-B); 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 also 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** method, 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 approach is compatible with *any* definition of target.

Focused crawlers have been studied in prior work [11], [12], [13], [14], [15]. Some aim to select some websites among thousands [13], as opposed to SDs within a given website; others aim at webpages on certain predefined topic [11], [12], [14], [15]. As we will show, traditional focused crawlers make

different assumptions, e.g., that all pages on a topic tend to be close within a website [16], and estimates page utility in a fundamentally different way, leading them to perform poorly on our problem (Sec. IV-E).

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 along 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 (no forms).

Contributions: Our contributions are as follows.

1) We formalize our **graph crawling problem** and show that optimally solving it is intractable (Sec. II).
2) We propose our **novel approach based on reinforcement learning (RL)** in Sec. III. 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. IV). Our crawler **outperforms all baselines**, including the closest equivalent from prior focused crawlers [14]. 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. V. For space reasons, the proof of Proposition 4, additional experiments, and a longer discussion on related work can be found in [17], along with open-source code to reproduce the experiments.

II. Problem Statement and Our Modeling

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

A. Graph Crawling Problem

We model a website as a rooted, node-weighted, edge-labeled directed graph. Each node represents a webpage, and each edge is a hypertext link leading from one to another. We fix a countable set $\mathcal{L}$ 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 encountered in a page.*

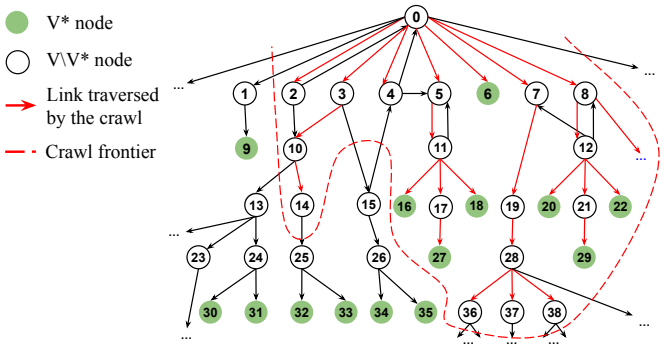


Fig. 1: Sample website, crawl, and frontier

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

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

The graph crawling problem is formalized as follows:

Problem 3. *Given a website graph* $G = (V, E, R, \omega, \lambda)$ *and a subset* V^* *of* targets *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* $k \in \mathbb{R}^+$*, determining whether there exists a crawl* $T = (V', E')$ *of* G *such that* $V^* \subseteq V'$ *and* $\omega(T) \leqslant k$ *is NP-complete; hardness holds even when* ω *and* λ *are constant functions.*

This is the decision variant of the graph crawling problem. Hardness is by reduction from the set cover problem [18]. 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*.

B. 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.

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 [19], [20], 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://ieee-icde.org/2025/ 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 [8], [9], **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 header requests. Depending on application needs, the crawler may be parameterized with any other MIME type set.

To each **edge** e, the edge labeling function λ associates a **label** $\lambda(e)$, derived from the HTML page's DOM structure [21], the standard tree-based representation of an HTML 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 the same 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 data volume the crawler receives with $\omega(u)$ as the page size, which is, in theory, unbounded. We also account for the cost c of *determining if a vertex is in* V^*, typically 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 $\omega(u)$. To reduce cost of performing HEAD requests, Sec. III-D discusses an alternative MIME type prediction method, resulting in a small, amortized c cost, which we can view as constant.

III. Crawling based on Reinforcement Learning

The goal of our efficient crawling is to retrieve as many targets as possible, at a minimum cost. Recall that we assume no prior knowledge of the website size or structure, thus in particular no knowledge of where the targets are. We hypothesise that *the label of an edge leading to a page* (derived from the DOM path leading to a hyperlink within its page) *can be used to learn which edges leads to target-rich pages*, and which edges do not. To obtain then leverage such insights, we need to both *explore* the website, to find promising paths, and *exploit* the knowledge thus gained, in order to retrieve targets. Thus, our approach to focused crawling is inspired by *reinforcement learning* (RL, in short) [22]. Below, we present the environment (states and actions) of our crawling task (Sec. III-A and III-B), and the learning algorithm (also called the *learning agent*) that evolves in this environment (Sec. III-C). We describe the URL classifier we use to compute the agent's rewards (Sec. III-D), and the overall crawling algorithm (Sec. III-E).

A. Environment: States and Actions

In reinforcement learning, 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*, which is a mapping from states to action, that maximizes the total reward. This model is known as a *Markov Decision Process* (MDP) [23].

Let us consider what we could use as *state* in our crawler. As the crawl starts from the root and visits other pages, one could think of using "the currently crawled webpage" as state. However, this choice is not suitable: on one hand, the current webpage does not reflect previous choices (the webpage may be accessible via several navigation paths). Also, an efficient crawler should not visit any webpage twice; in contrast, learning supposes multiple visits to a state, to refine and then leverage information learned in that state. Therefore, we use a **single-state model**, where the agent is perpetually in exactly only one state. What matters then is only the set of actions that the agent can follow at each step of the crawl. Intuitively, an *action* is to navigate along some link.

B. Grouping Links into Actions

Following our hypothesis that similar links have similar values (in terms of likelihood of leading to targets), we model an action as *a group of similar links*, with the semantics that taking the action amounts to crawling along these links. As explained above, we *label each link based on the DOM path leading to the link in its enclosing page*; we therefore need a **measure of similarity between paths**. To that effect, we first represent each DOM path by a *numerical bag-of-word vector* p, over the n-grams vocabulary built from all DOM paths encountered so far. We rely on n-grams which preserve the order in which HTML node names appear in a path, because in our context, this order is significant (as Sec. IV confirms). Because the n-grams vocabulary is dynamically built during the crawl, bag-of-word vectors assigned to links encountered at different times have different lengths. To be able to compute distances between our paths' representations, we first *project each path into a vector of a fixed dimensionality* D, where $D = 2^m$ for a chosen $m > 0$. We do this as follows.

1) We use a *hash function* $h : x \mapsto \left\lfloor \frac{(\Pi \times x) \mod 2^w}{2^{w-m}} \right\rfloor$ which maps any integer x to a number between 0 and $D - 1$. Its parameters Π (a large prime number) and w are fixed, and chosen such that $w > m$.
2) Calling h on each position betwen 0 and $d - 1$, where d is the number of positions in the bag-of-words vector p,

[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.

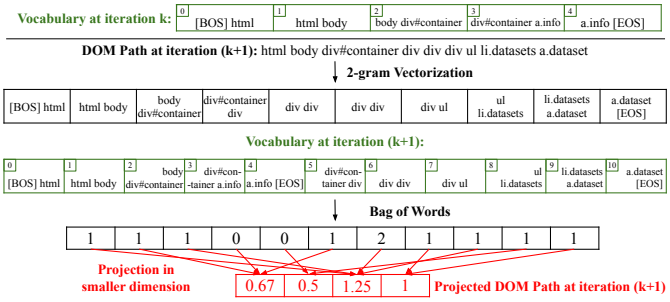


Fig. 2: Mapping of a DOM path into a vector, using n=2.

maps that position somewhere between 0 and $D-1$. This enables us to transform 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 in particular at the beginning of the crawl, when the set of HTML element names seen so far is small (thus d is small).

Figure 2 illustrates the projection of a DOM path via 2-grams, into a 4-dimensional vector (BOS, EOS denote begin/end of stream). This also adds to the 2-grams vocabulary.

Our next task is to **find, for each projected link representation p_D, the nearest action** a_{closest} among all known actions A. Conceptually, we see an action as an **evolving set (cluster)** of similar links, together with a **centroid**, which is the mean of all projected links. For efficiency, as we will show, for each action, we only store the centroid, which changes as the action's set of links evolves.

Algorithm 1 computes, for each p_D, its nearest action, if it is close enough to p_D; otherwise, a new action will be created and returned. For that, we insert the action centroids in a *Hierarchical Navigable Small Worlds* (HNSW) [24] index denoted $\mathcal{I}$. We chose HNSW because updating it is highly efficient, and we need to update an action's centroid each time a new link joins the action. While maintaining and querying such an index might be costly in CPU time, this cost is negligible compared to the crawl time (either waiting between two requests, or for the download of webpages). The index also gives an estimation of the *cosine similarity* between p_D and its closest centroid. If this similarity is above a **threshold** θ, the link joins the action. Otherwise, we create a new action composed of p_D only. As our experiments show (Sec. IV), the choice of the threshold has a significant impact on the performance of our learning agent. As an extreme case, if the threshold is 0, all links are grouped in a single action, and the agent cannot learn anything; it will randomly select the links to follow. On the other hand, if the threshold is 1, each link makes up an action. As a result, the agent only explores, and never exploits, since no action can be repeated. A useful threshold should enable enough actions to separate interesting groups of links from non-interesting ones, while still allowing learning and exploitation.

Algorithm 1: Finding the action for a hyperlink

Input: Action set A, projected DOM path p_D
Output: Action a

$a_{\text{closest}} \leftarrow$ Approx. nearest neighbor (from HNSW index $\mathcal{I}$) of p_D;
$d_{\text{closest}} \leftarrow$ Cosine distance between a_{closest} and p_D;
if $1 - d_{closest} \geqslant \theta$ **then**
  Update centroid of a_{closest} 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;
  $a_{\text{closest}} \leftarrow a_{\text{new}}$;
return a_{closest}

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.

C. Learning Agent: Sleeping Bandit

The family of reinforcement models dedicated to stateless (one-state) environments are called *Multi-Armed Bandits* (MABs) [22]. The standard, deterministic, MAB algorithm that optimizes the trade-off between exploration and exploitation is the *Upper-Confidence Bound* (UCB) [25]. This method is based on the principle of *optimism under the face of uncertainty*. A score is computed for each action at every time step, and the action with the maximal score is chosen. The MAB score of action a at time $t+1$, denoted $s^{t+1}_{\text{MAB}}(a)$, includes an **exploitation term**, which is the mean of rewards obtained so far for that action, and an **exploration** term, reflecting the number of times the action has been selected, with respect to the total length of the crawling history:

$$s^{t+1}_{\text{MAB}}(a) = R^t_a + \alpha\sqrt{\frac{\log(t)}{N_t(a) + \varepsilon}}$$

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 action a has been selected up to time t, and ε is a small value to cope with the case where we have not selected action a yet.

Once all the URLs from an action have been visited, given that we never visit the same URL twice, an action may become empty. [26] presents an alternative to UCB called *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 unvisited links in action a at time-step t, and 0 otherwise.

Once an action a is selected, our crawler choses *randomly among the unvisited links* from a the link l to traverse next. This ensures all links have an equal chance to be selected.

How should we reward the crawling along $l \in a$? Considering that our goal is to find *new targets*, the reward should reflect the number of *not-yet-discovered links to targets*, contained in the HTML page h reachable by following l. This "novelty" condition is set to help the learning agent focus on unseen targets. For instance, if h turns out to contain 12 links, only 5 of which lead to targets, and such that we had already retrieved 2 of these targets during the crawl, the reward for following l to h should be 3. This simple choice gives good results, as shown in Sec. IV.

However, a problem remains to be solved: we need to **compute the reward of all links, before following any link** in h. Following a link is an action itself, adding to the cost of our crawl, thus we need a (cheap) *estimation*. Further, we need this estimation for any link, even though (depending on how the crawl evolves) some links in h may actually *never* be followed. For this, we introduce a *URL classifier*, which, just by inspecting the links in the newly acquired page h, can compute the reward. This classifier is *online*, meaning it evolves during the crawl, and it is crucial to the performance of our approach. We detail it in Sec. III-D.

Lastly, we discuss the value of the parameter α in our learning agent. UCB and AUER are known to be optimal when $\alpha = 2\sqrt{2}$. However, this holds only in their standard configuration, where the reward is normally distributed in $[0, 1]$. In contrast, the values of our reward (counting new targets) are unbounded. Further, it is highly unlikely that the distribution of our reward is normal. In fact, we *hope* this is *not* the case: it would mean that a majority of HTML pages contain links to targets. If this held, we would have to visit the entire website, or at least a vast share, to retrieve most targets. Setting up α to guarantee optimality in our case would require knowing the shape of the distribution of rewards, which is not possible. Therefore, we keep $\alpha = 2\sqrt{2}$, which gives good results in practice, across very varied reward distributions (as shown in Sec. IV-A, in particular in Figure 3).

D. Estimating Rewards with an Online URL Classifier

Determining the reward of an action taken by our agent requires to assess the MIME type of a page, from its URL. One way to do so is to make an **HTTP HEAD** request to the website; this remote call remains expensive, and they should be spaced, e.g., by 1 second, for crawling ethics. Using the extension (last URL segment), e.g., `.html`; `.pdf`, `.csv`, etc., fails on links such as https://www.justice.gouv.fr/en/node/9961, which has no extension; the same happens in other websites, e.g., ILO, etc. To be efficient and generic, we devise an **online URL classifier to estimate rewards**, as follows.

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

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;
  $\mathcal{C}$ is incrementally trained on batch (X_{mat}, y);
  $(X, y) \leftarrow (\emptyset, \emptyset)$;
  initial_training_phase $\leftarrow$ False;
if initial_training_phase **then**
  $l_U \leftarrow$ The class of the MIME type obtained from HTTP HEAD request on U;
  Add (U, l_U) to (X, y);
  return l_U
else
  $U_{\text{vec}} \leftarrow$ 2-gram bag-of-words vector of U;
  return $\mathcal{C}(U_{\text{vec}})$

We initially set our classifier to assign one of these three classes to each link found in a newly crawled page h. However, the classifier proved unable to predict, based on the URL, pages that result in a 4xx or 5xx errors, from those that resolve fine. Intuitively, this is because the former URLs are often very similar to URLs of "HTML" or "Target" pages that are accessible. However, *different classification errors do not have the same impact on crawling*:

- Misclassifying a "Neither" URL as "HTML" or "Target" only incurs the extra cost of following the URL; (immediately in the second case, later in the first case); this cost is moderate, since it will likely throw an error.
- 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 last case can make the crawler miss huge parts of the website.

To avoid the second kind of errors completely, our classifier is trained on just **two classes ("HTML" and "Target")**, even though we know some URLs are neither.

Algorithm 2 shows how we train and then exploit our classifier $\mathcal{C}$. $\mathcal{C}$ is a logistic regression [27] classifier, iteratively trained through epochs of *Stochastic Gradient Descent* (SGD) [28], with batch size b. The input to the classifier is a vector of character-wise 2-grams of the original URL. For instance, the URL https://www.A.com/data/file.csv is transformed into a list $[\text{ht}, \text{tt}, \text{tp}, \ldots, \text{.c}, \text{cs}, \text{sv}]$. We associate to each pair of usual ASCII characters[2] a unique identifier, and encode each URL as a bag-of-words over this vocabulary.

As Algorithm 2 shows, during the first training epoch, we use HTTP HEAD requests to label a number of b URLs; let X denote the set of URI and y their set of labels. Based on this,

[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;
    $N_t(a_c) \leftarrow N_t(a_c) + 1$;
  else
    $u \leftarrow$ Select a link from $\mathcal{F}$ uniformly at random;
  crawl_next_page(u) (Algorithm 4);

we compute rewards, and assign URLs to either the frontier $\mathcal{F}$ or the set of targets V^*. After this epoch, we set the Boolean initial_training_phase to false, and the classifier is used to infer URL classes, without making new HTTP HEAD requests. We further improve our classifier by training it *online*: **during execution, any HTTP GET request issued by our crawl contributes an annotated (URL, class) pair**, which we add to our classifier's X and y for incremental training when reaching the batch size.

In this way, after the first training epoch, getting labels has no extra cost. Since we both want to use the classifier as soon as possible and train it often to improve its accuracy, we have to set b relatively small. This online method is able to **adapt to a potential change in the form of the URLs**, e.g., when the crawl discovers new parts of the website where URLs are formatted differently. An off-line method would not be able to adapt, even if trained on many examples initially.

E. Crawling Algorithm

We now present the overall crawling procedure, described in Algorithm 3. Initially, the tree T of already crawled pages, the set A of actions and the frontier $\mathcal{F}$ are empty. The **budget** β spent by the crawler up to this point, and the crawling step (or time) t are both set to 0. The crawl starts by crawling the original link r. It terminates either when there is no new link to be followed (the website is completely crawled) or if the maximum budget B has been reached. Then, if the set of actions is not empty, the learning agent choses its action a_c for this crawling-step, following the Sleeping-Bandit procedure described in Sec. III-C. Among the links in $\mathcal{F}$ associated with action a_c, the agent choses a link uniformly at random. Then, it crawls the page behind this link, using Algorithm 4.

Algorithm 4 starts by retrieving several information from the response of the HTTP GET request: the HTTP status, the MIME type extracted from the header, and the body (content) of the webpage. The cost of that request is added to the budget β. The HTTP response status may fall in one of three cases, as shown in the algorithm: 1) Errors (on the client or server side) correspond to **4xx** or **5xx** statuses. Such an error leads neither to a new link nor to a target. 2) HTTP statuses of the form **3xx** are redirections. In this case an extra "Location" header is provided, with the URL of the redirection: we follow it and crawl it, if we had not done this before. 3) A **2xx** status means the server successfully sends a response; it is either an HTML page, a target (that is, a page whose MIME type is in the list L presented in Sec. II-B), or neither. In these first two cases, X and y are updated. For HTML pages, new hyperlinks U_{new} are extracted from the HTML code. Only URLs within the website are kept.

For each link, if the corresponding URL is not already in the frontier or in T, the URL is assigned a class of by calling Algorithm 2. If it points to a HTML webpage, we map the link to an action using Algorithm 1. The set of actions A is updated

Algorithm 4: Crawl next page

Input: u the URL to be crawled
Add URL u to T;
$t \leftarrow t + 1$;
http_response, mime_type, body $\leftarrow$ Result of the request on URL u with HTTP GET;
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);
      $U_{\text{new}} \leftarrow$ All hyperlinks in body pointing to the same website as r;
    else if mime_type $\in L$ **then**
      Add (u, "Target") to (X, y);
      Add target (the content of body) to V^*;
    return;
else if http_response *is 3xx (redirection)* **then**
  $u \leftarrow$ location from result of the request on URL u with HTTP GET ;
  if $u \notin T \cup \mathcal{F}$ **then**
    crawl_next_page(u) (recursive call);
    return;
reward $\leftarrow 0$;
for $u_{\text{new}} \in U_{\text{new}}$ **do**
  if $u_{\text{new}} \notin T \cup \mathcal{F}$ **then**
    $l_{u_{\text{new}}} \leftarrow$ 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}) (recursive call);
      reward $\leftarrow$ reward $+ 1$;
if $u \neq r$ **then**
  $R_{\text{mean}}(a_c) \leftarrow R_{\text{mean}}(a_c) + \frac{\text{reward} - R_{\text{mean}}(a_c)}{N_t(a_c)}$

(an action's centroid changes, or a new action is added), and the chosen action a_c is mapped to the link, which is then added to frontier $\mathcal{F}$. If a target is behind the link, we immediately acquire it by making a recursive call to Algorithm 2, and update the reward. Finally, if the page crawled by Algorithm 2 is not the starting page, the mean reward of chosen action a_c is updated, to reflect the last computed reward.

IV. Experimental Results

We now present our experimental results: websites used (Sec. IV-A), poor performance of search engines for this task in Sec. IV-B, baselines in Sec. IV-C, comparison of our crawler to baselines in Sec. IV-E, and impact of hyper-parameters in Sec. IV-F. Sec. IV-G examines our Sleeping Bandit (SB) algorithm's effectiveness in grouping links into coherent and different-reward group, and Sec. IV-H analyzes the quality of our URL classifier, before concluding our experiments.

A. Websites

Table I depicts 18 websites on which we test our crawler. As our primary application is a collaboration with French journalists [9], six of these websites are in French, public, and governmental. Each contains material about a branch of government, including, but not limited to, statistics (which our crawler targets). We also include eight multilingual websites: the UN's International Labor Organization (*il*), the French official statistical institute (*is*), Japan's Ministry of Interior (*jp*), the Organization for Economic Co-operation and Development (OECD, *oe*), the Open Knowledge Foundation (*ok*), Qatar's official statistical service (*qa*), the UN World Health Organization (*wh*), and the World Bank (*wo*). Finally, we add national websites in English: the Australian Bureau of Statistics (*ab*), the US National Center for Education Statistics (*nc*), the US Census (*ce*), and the US Bureau of Economic Analysis (*be*). Websites with pages in at least two different languages have a ✓ in the "**Mlg.**" column.

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. IV-D).

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. IV-D). "**#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 the root to the target. Depths vary greatly, from less than 1 to almost 90, both in mean and variance. Highest average depths arise on portals that require page-to-page navigation, such as *ju*, hosting the *Bulletin Officiel* publication over several years.

Table I shows out websites are extremely 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.

B. Search engines and dataset search

We report here our experiments finding SDs via search engines (SEs). We focus on three popular Google services: classical search[3] (GS), *Google Datasets Search*[4] (GDS), and the *Google Public Data Explorer*[5] (GPDE) providing open data from international organizations, whose goal is comparable with our statistical data journalism application.

GS allows filtering by website (site: operator) and file type (filetype:), either on its Web interface or through an API[6]). However, Google (like other SEs) typically **does not fully index any website**, and it **limits search results to 1 000**, which may drastically truncate them. For instance, querying *ju* for PDFs yields only 302 results, while 9 000 exist. This issue is consistent across tested websites and MIME types: on *ju* we only get 240 ODS files (out of 910), and on *in* 38 XLS out of 1 546. Even worse, GS doesn't recognize the TSV file type, of which *ju* holds 11 097. On *il*, only 641 results are listed instead of (at least) 49 962, etc.

GDS **trims results even further**, e.g., only 109 tabular files in *ju*, out of 1 188. For *il*, 93 datasets are returned (our crawler finds more than 170 000), but GDS is not able to detect their associated MIME types. For *ce*, GDS returns 312 datasets, while we find more than 800 000, etc.

GPDE is built for human website readers. It indexes SDs from a **closed, fixed set of providers**, including Eurostat, World Bank, Canadian Statistical Institute, and *oe*. Important sites, such as UN's *wh* and *il*, or US sites such as *be*, *nc*, and *ce* are not present. **GDPE only allows manual exploration of its datasets**, and plots statistic curves according to users' filters; there is **no download** facility and **no pointer to the original dataset**. For instance, GPDE allows inspecting 150 OECD statistics over 37 countries and 100 years, yet the only outside reference is a a 272 pages PDF, in the OECD iLibrary.

Overall, SEs do not offer **transparency or control over the selected SDs**. This is an important limitation, especially for data journalism applications. In contrast, our crawler retrieves all data files within a specified budget, with explainable decisions, and a choice over which MIME types to target.

C. Baselines

First, we consider four simple baselines. (i) The *random crawler* selects the next hyperlink to visit uniformly at random

[3]https://www.google.com/
[4]https://datasetsearch.research.google.com/
[5]https://www.google.com/publicdata/directory
[6]https://programmablesearchengine.google.com/

TABLE I: Main characteristics of websites (see detailed crawling methodology in Sec. IV-D)

	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)

from the crawl frontier. (ii) The *Breadth-First Search (BFS)* exhaustive crawler maintains the frontier as a *First-In-First-Out queue* data structure. It crawls all pages reachable by a path of length ℓ before any page reachable only by longer paths ($\ell' > \ell$). (iii) *Depth-First Search (DFS)* maintains the frontier in a *Last-In-First-Out stack*. It is rarely used for exhaustive crawling due to its susceptibility to robot traps. (iv) The unrealistic *Omniscient Crawler* knows all target URLs (V^*) from the start and crawls them one after the other. Since an optimal crawler is intractable (Prop. 4), this omniscient crawler provides an (unreachable) upper bound on crawler efficiency.

The *Offline-Trained, DOM-Based Crawler (DOM-OFF)* [14] is a close competitor from the literature. It seeks to retrieve *diverse* textual content from websites. It initially crawls a fixed part of the website (3 000 pages) using BFS, and groups the DOM paths of followed links as we do in Sec. III-A. The benefit of a page is computed immediately; DOM path groups are stored in a priority queue, ordered by average benefit. In the crawl after 3 000 pages, [14] *only considers new links fitting one of the DOM path groups* (ordered by the group priority); other links are ignored. In our context, page benefit computation requires knowing whether its links lead to targets, thus traversing the link. To enable this baseline, we provide it with the true benefit (number of targets behind a page's links) freely, as if given by an oracle, for the first 3 000 pages. After 3 000 pages, new DOM path groups can form, but they are assigned a permanent benefit of zero. This baseline, given an unfair advantage in its first stage, can be seen as an *ablated* version of our crawler, which learns *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. This classifier is used to update the frontier as a *priority queue*, hopefully giving priority to targets. It employs a standard *logistic regression* that is periodically trained on crawled webpages (thus, at no extra HTTP request cost). It takes as input standard focused crawler features, as in, e.g., [11], [29], [15]: the (approximated) depth of the page were the link was found, the 2-gram vector representation of the hyperlink's URL (similarly to Sec. III-D), and the 2-gram vector representation of the hyperlink's *anchor* (the text associated with it). We put aside features that are *topic-oriented* (see Sec. V for a discussion on the differences between our problem and traditional focused crawlers). This baseline can also be seen as an *ablated* version of our crawler, which neither benefits from HTML link structure information, nor from reinforcement learning.

D. Practical crawling in our experiments

To evaluate six baselines and our crawler with various hyper-parameters, repeated crawls over each website are not feasible, due to (a) time constraints (especially with large files and/or slow connections), and (b) crawling ethics.

Therefore, for each website from our dataset, we first launched the baselines and our crawler, in a setting where each crawler first checks if the resource was not already fetched in a local database. If it was, we use it. Otherwise, we request it via HTTP GET and save the result in the local database (URL, HTTP status, headers, and response body).

For evaluation purposes only, we stop crawling on a given website, as follows: (i) If one of the crawlers terminates before crawling **1M** pages, then the entire website was crawled, and we have a complete, local replication; all other crawlers are then stopped, and evaluated just using local database look-ups. (ii) If each crawler visited 1M webpages (possibly not the same pages). (iii) If some crawlers are still running after a time limit. Crawling can be extremely slow for some websites; in our study, we set a practical limit at **3 weeks**. Doing so on our set of 18 websites for each crawler 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 having *less than 1M pages, fully-crawled*. The full replication facilitates running crawlers with multiple parameter settings. On these websites we also **gather**

complete knowledge for the (unrealistic) SB-ORACLE crawler (see Sec. IV-E). (ii) On websites where *all baselines could crawl 1M webpages*, crawlers are compared **on the subset of the website each of them crawled**. Note that their sets of targets, and the volume of both non-target and target data retrieved, may differ; a good crawler selects a *more target-rich webpage set*. (iii) For websites where *at least one baseline did not crawl 1M pages within the time limit*, we compare crawlers **on the smallest number of pages** visited by one of them (the first, for those who crawled more). For instance, if three crawlers visit, respectively, 400k pages, 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.

E. Comparison with Baselines

Figure 3 compares our crawler's performance to the baselines. Due to space limitations, we only present a selection of 10 websites (exhaustive plots appear in [17]). For each crawler, and all 18 websites, Table II shows the percentage of queries needed to retrieve 90% of the targets, and Table III 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 I). When a crawler never reaches the desired 90%, we show $+\infty$.

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. IV-D). All baselines but the random one are deterministic, thus one run suffices.

The plots and measures in Tables II and III (where 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 crawlers 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 [14], 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 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. IV-H.

F. 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. Full graphical results are available in our technical report [17], while Table IV provides a summary. When varying one hyper-parameter, others keep their default values (see Sec. IV-E).

1) *Impact of α.* α controls the trade-off between exploration and exploitation (Sec. III-C). Table IV (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. III-A), 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 IV (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.

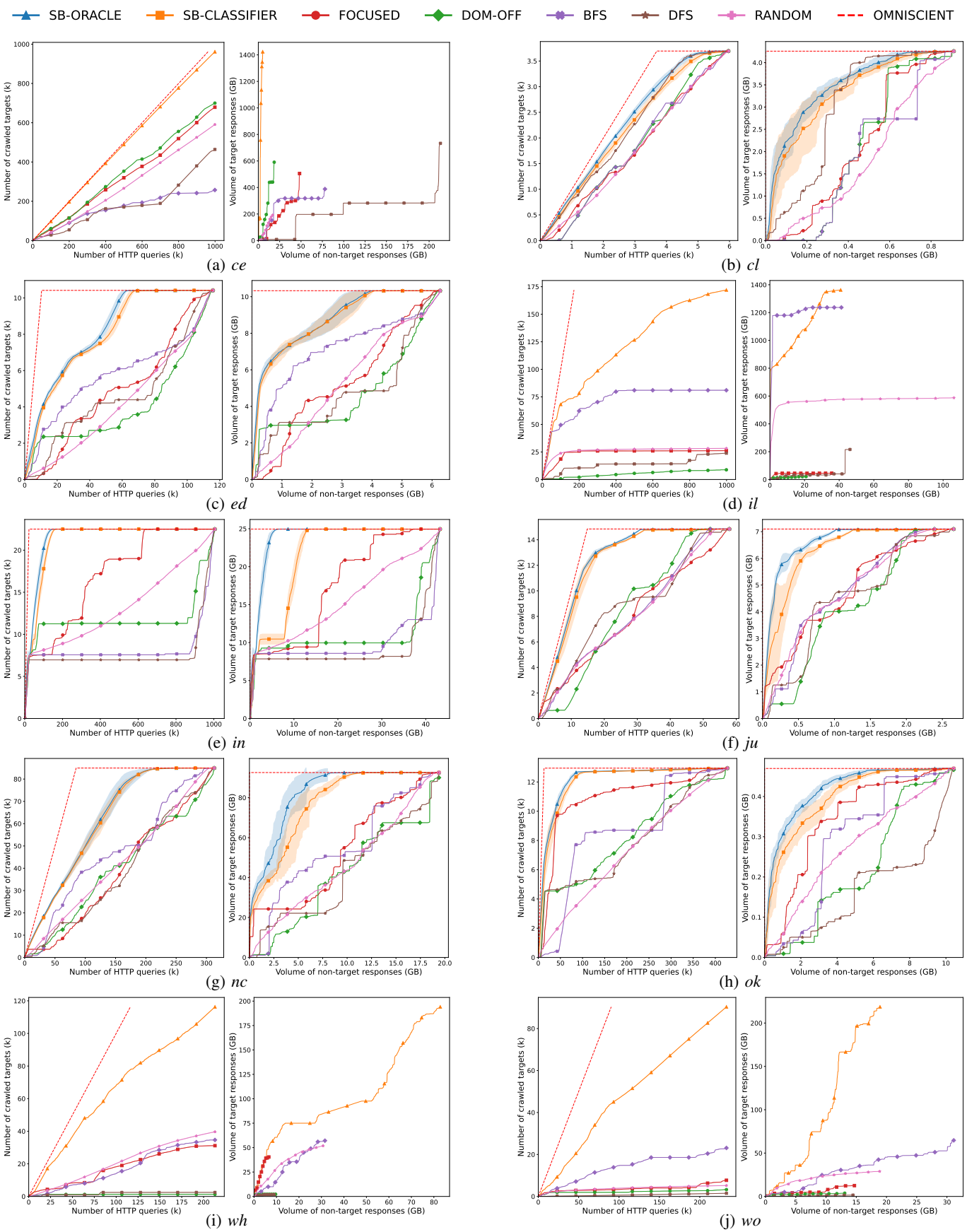


Fig. 3: Comparison of different crawler performance for 10 selected websites presented in Table I

TABLE II: Percentage of queries that each crawler performs to retrieve 90% of the targets, for all websites.

Crawler	*ab*	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*
SB-ORACLE	NA	NA	72.6	NA	70.6	70.3	47.9	NA	12.8	73.8	NA	34.1	50.7	55.8	13.8	48.7	NA	NA
SB-CLASS.	**44.6**	**35.1**	**83.9**	**23.5**	74.6	**70.1**	**52.0**	**14.2**	**14.9**	**76.7**	**38.5**	**35.8**	**51.8**	**59.2**	**16.0**	**60.2**	**20.1**	**20.3**
FOCUSED	68.2	$+\infty$	87.7	36.0	88.8	82.7	86.5	$+\infty$	62.8	86.9	41.5	91.1	92.7	84.9	51.8	73.2	$+\infty$	$+\infty$
DOM-OFF	96.4	50.3	89.2	34.7	81.6	88.2	95.6	$+\infty$	99.7	88.0	$+\infty$	74.3	94.0	88.7	76.2	84.7	$+\infty$	$+\infty$
BFS	97.4	90.8	89.1	73.4	87.4	80.0	94.6	33.2	99.3	92.7	46.2	80.8	81.6	96.5	66.9	70.8	79.0	92.0
DFS	83.7	$+\infty$	85.2	74.9	**70.5**	84.6	90.4	$+\infty$	99.7	87.7	45.3	80.2	93.7	88.7	80.5	75.2	$+\infty$	$+\infty$
RANDOM	$+\infty$	98.2	92.4	44.5	89.1	85.1	94.9	$+\infty$	98.9	92.7	$+\infty$	83.2	87.8	96.8	85.0	79.1	71.0	$+\infty$

TABLE III: Fraction of non-target volume retrieved by each crawler, before retrieving 90% of the total target volume.

Crawler	*ab*	*as*	*be*	*ce*	*cl*	*cn*	*ed*	*il*	*in*	*is*	*jp*	*ju*	*nc*	*oe*	*ok*	*qa*	*wh*	*wo*
SB-ORACLE	NA	NA	24.2	NA	56.3	24.6	49.2	NA	12.5	59.7	NA	22.9	29.5	48.0	33.2	33.9	NA	NA
SB-CLASS.	**63.1**	**21.4**	**41.8**	**41.5**	57.6	**29.2**	**49.9**	55.5	**27.4**	83.5	**39.7**	**27.9**	**38.2**	**49.5**	**38.9**	**39.0**	**17.4**	**55.5**
FOCUSED	$+\infty$	$+\infty$	85.2	97.0	76.3	74.7	86.4	$+\infty$	67.3	**73.8**	66.5	72.2	84.9	72.7	49.8	80.3	$+\infty$	$+\infty$
DOM-OFF	$+\infty$	$+\infty$	92.6	64.4	65.0	94.7	92.9	$+\infty$	98.8	89.7	$+\infty$	72.3	88.8	89.0	73.6	51.0	$+\infty$	$+\infty$
BFS	81.8	75.7	66.5	98.5	80.8	50.4	93.2	**3.6**	98.9	93.8	53.6	68.0	84.5	97.5	63.3	86.6	91.5	98.3
DFS	98.6	$+\infty$	64.2	97.0	**45.0**	82.4	90.8	$+\infty$	98.1	85.0	59.1	68.6	96.1	90.5	97.1	79.9	$+\infty$	$+\infty$
RANDOM	71.6	$+\infty$	83.4	$+\infty$	89.3	82.7	92.9	$+\infty$	95.8	98.3	$+\infty$	70.1	88.2	98.1	86.6	93.6	$+\infty$	$+\infty$

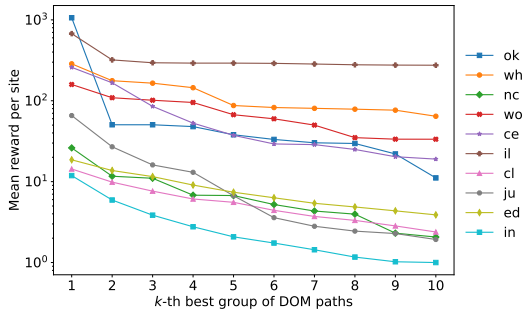


Fig. 4: Mean rewards of the top-10 groups of DOM paths for the websites selected in Figure 3 (log y-scale).

3) *Impact of* θ. The similarity threshold (Sec. III-A) affects how DOM paths are grouped. We tested low (0.55), high (0.95), and well-performing (0.75) values. Table IV (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.

G. 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 each of the selected websites in Figure 3. 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, 67 for fourth, etc., down to 41 for the 10th group. This shows that **our Sleeping Bandit 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 V 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* it is more than 20 times its mean. This supports our pragmatic choice of $\alpha = 2\sqrt{2}$, which proves effective in practice.

H. URL Classification Quality

The confusion matrix of our URL classifier, averaged over 15 runs, for all fully-crawled websites, is presented in Table VI. Confusion matrices at the website granularity can be found in the technical report [17]. We see that **classification errors are extremely marginal on "HTML" and "Target" URLs**. Not classifying as "Neither" leads to some classification errors (discussed in Sec. III-D), 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 classifier oracle is unfeasible in practice, the performance of SB-CLASSIFIER is satisfactory.

I. Conclusions

Our crawler based on reinforcement learning 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**. It delivers high fractions of each sites (volume of) targets while crawling only a small part.

TABLE IV: 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**

TABLE V: 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

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

True/Predicted	**HTML (%)**	**Target (%)**	**Neither (%)**
HTML	58.04	1.37	0
Target	0.75	32.19	0
Neither	5.34	2.41	0

V. Related Work

Focused crawlers [11], [29] 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. [12] 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 Sleeping Bandit model, as not all actions are always usable. [15] brings RL into focused crawling through an MDP [23] with new state and action representations. We differ by using a single-state model as detailed in Sec. II-B, and, again by focusing on page types instead of topics. A non-topical focused crawler aimed at content-rich pages is detailed in [14], with an initial training phase followed by intensive exploration. Sec. IV-E shows that in our setting, this underperforms compared to our Sleeping Bandit approach. [13] describes a focused crawler for data-rich webpages, using MABs. Their focus is on selecting optimal *websites* from thousands, contrasting with our focus on finding specific *pages* within websites. The two could be combined to automatically identify targets in a large collection of websites.

Incremental or *revisit* crawlers [30], [31], [32], [33] focus on revisiting already crawled websites looking for new content. More recent works leverage machine learning to do so: [34], [35], [36] study effective *predictors* (features) of new outlinks in previously visited websites, while [37], [38] 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* [39] and *distributed* [40], [41], [42] crawling of Web and social networks, or as Web forums [43], [44]; *hidden-* or *deep-Web* crawlers [45] aim for content behind Web forms. These aspects are mostly 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 in the context of Web crawling. [46], [47] use the website structure to cluster webpages, within *Web scrapers* which extract and structure data from HTML pages. [14] directly uses paths in the DOM of each HTML page in order to do focused Web crawling.

While UCB [25] is the state of the art for MABs, simpler approaches like ε-greedy and its variants [22] randomly select actions with probability ε and otherwise choose the highest scoring action. We also considered Thomson Sampling (TS) [48], a probabilistic method based on posterior distribution estimation. We excluded both to ensure our crawler's *stability*; also, TS requires prior distributions, unavailable to us due to lack of prior knowledge about the websites.

VI. Conclusion

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

In our future work, we would like to explore more complex reward functions than the simple number of targets reachable. Also, integrating deep-web crawling techniques could enhance our crawler's ability to access data behind forms or within portals more efficiently, thereby further improving our web data acquisition at scale. Finally, adapting our crawler to *incrementally revisit* sites focusing on new targets is a natural extension, especially for statistical data journalism, as new statistics are regularly published.

REFERENCES

[1] C. Lockard, P. Shiralkar, X. L. Dong, and H. Hajishirzi, "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*, J. Caverlee, X. B. Hu, M. Lalmas, and W. Wang, Eds. ACM, 2020, pp. 888–889. [Online]. Available: https://doi.org/10.1145/3336191.3371878

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives, "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*, ser. Lecture Notes in Computer Science, K. Aberer, K. Choi, N. F. Noy, D. Allemang, K. Lee, L. J. B. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds., vol. 4825. Springer, 2007, pp. 722–735. [Online]. Available: https://doi.org/10.1007/978-3-540-76298-0_52

[3] T. P. Tanon, G. Weikum, and F. M. Suchanek, "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*, ser. Lecture Notes in Computer Science, A. Harth, S. Kirrane, A. N. Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, Eds., vol. 12123. Springer, 2020, pp. 583–596. [Online]. Available: https://doi.org/10.1007/978-3-030-49461-2_34

[4] G. Weikum, X. L. Dong, S. Razniewski, and F. M. Suchanek, "Machine knowledge: Creation and curation of comprehensive knowledge bases," *Found. Trends Databases*, vol. 10, no. 2-4, pp. 108–490, 2021. [Online]. Available: https://doi.org/10.1561/1900000064

[5] "Sdmx technical specifications," https://sdmx.org/?page_id=5008, 2024.

[6] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer, "Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2508–2521, 2020. [Online]. Available: http://www.vldb.org/pvldb/vol13/p2508-karagiannis.pdf

[7] M. Saeed and P. Papotti, "Fact-checking statistical claims with tables," *IEEE Data Eng. Bull.*, vol. 44, no. 3, pp. 27–38, 2021. [Online]. Available: http://sites.computer.org/debull/A21sept/p27.pdf

[8] T. D. Cao, I. Manolescu, and X. Tannier, "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, 2018, pp. 4:1–4:6. [Online]. Available: https://doi.org/10.1145/3201463.3201467

[9] O. Balalau, S. Ebel, T. Galizzi, I. Manolescu, Q. Massonnat, A. Deiana, E. Gautreau, A. Krempf, T. Pontillon, G. Roux, and J. Yakin, "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, 2022, pp. 4798–4802. [Online]. Available: https://doi.org/10.1145/3511808.3557198

[10] O. Balalau, E. Simon, H. Galhardas, T. Galizzi, and I. Manolescu, "STaR: Space and Time-aware Statistic Query Answering," in *CIKM 2024 - 33rd ACM International Conference on Information and Knowledge Management*, Oct. 2024.

[11] S. Chakrabarti, M. van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific Web resource discovery," *Computer Networks*, vol. 31, no. 11, pp. 1623–1640, 1999.

[12] G. Gouriten, S. Maniu, and P. Senellart, "Scalable, generic, and adaptive systems for focused crawling," in *Proceedings of the 25th ACM Conference on Hypertext and Social Media*, 2014, p. 3545.

[13] R. Meusel, P. Mika, and R. Blanco, "Focused Crawling for Structured Data," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, 2014, p. 10391048.

[14] M. Faheem and P. Senellart, "Adaptive web crawling through structure-based link classification," in *Proc. ICADL*, 2015, pp. 39–51.

[15] M. Han, P.-H. Wuillemin, and P. Senellart, "Focused Crawling Through Reinforcement Learning," in *Web Engineering*. Springer International Publishing, 2018, pp. 261–278.

[16] B. D. Davison, "Topical locality in the Web," ser. SIGIR '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 272279. [Online]. Available: https://doi.org/10.1145/345508.345597

[17] A. Gauquier, I. Manolescu, and P. Senellart, "Efficient Crawler for Scalable Web Data Acquisition (technical report and experiment reproducibility kit)," https://github.com/AntoineGauquier/efficient_crawler_for_scalable_web_data_acquisition, 2024.

[18] D. S. Johnson and M. R. Garey, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979, ch. A3.1.

[19] P. Senellart, "Identifying Websites with Flow Simulation," in *Web Engineering, 5th International Conference, ICWE 2005, Sydney, Australia, July 27-29, 2005, Proceedings*, ser. Lecture Notes in Computer Science, D. B. Lowe and M. Gaedke, Eds., vol. 3579. Springer, 2005, pp. 124–129. [Online]. Available: https://doi.org/10.1007/11531371_18

[20] A. Alshukri, F. Coenen, and M. Zito, "Web-Site Boundary Detection," in *Advances in Data Mining. Applications and Theoretical Aspects, 10th Industrial Conference, ICDM*, vol. 6171, 2010, pp. 529–543.

[21] WHATWG, "DOM: Living standard," https://dom.spec.whatwg.org/, 2024.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. The MIT Press, 2018, ch. 1, 2.

[23] R. A. Howard, *Dynamic programming and markov processes*. John Wiley, 1960.

[24] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[25] P. Auer, N. Cesa-Bianchi, and P. Fischer, " Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, 2002.

[26] R. Kleinberg, A. Niculescu-Mizil, and Y. Sharma, " Regret bounds for sleeping experts and bandits," *Machine Learning*, vol. 80, no. 2, pp. 245–272, 2010.

[27] G. James, D. Witten, T. Hastie, R. Tibshirani *et al.*, *An introduction to statistical learning*, 2013, vol. 112, ch. 4.3.

[28] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of 19th International Conference on Computational Statistics*, 2010, pp. 177–186.

[29] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, M. Gori *et al.*, "Focused Crawling Using Context Graphs," in *VLDB*, 2000, pp. 527–534.

[30] J. Cho and H. Garcia-Molina, "Effective page refresh policies for Web crawlers," *ACM Trans. Database Syst.*, vol. 28, no. 4, p. 390426, Dec. 2003. [Online]. Available: https://doi.org/10.1145/958942.958945

[31] ——, "The Evolution of the Web and Implications for an Incremental Crawler," in *VLDB*, vol. 2000, 2000, pp. 200–209.

[32] K. Sigurðsson, "Incremental crawling with Heritrix," 2005.

[33] G. Mohr, M. Stack, I. Rnitovic, D. Avery, and M. Kimpton, "Introduction to Heritrix," in *4th International Web Archiving Workshop*, 2004, pp. 109–115.

[34] T. K. N. Dang, D. Bucur, B. Atil, G. Pitel, F. Ruis, H. Kadkhodaei, and N. Litvak, "Look back, look around: A systematic analysis of effective predictors for new outlinks in focused Web crawling," *Knowledge-Based Systems*, vol. 260, p. 110126, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705122012229

[35] K. Avrachenkov, K. Patil, and G. Thoppe, "Online algorithms for estimating change rates of web pages," *Performance Evaluation*, vol. 153, p. 102261, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016653162100078X

[36] A. Kolobov, Y. Peres, E. Lubetzky, and E. Horvitz, "Optimal freshness crawl under politeness constraints," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR'19. New York, NY, USA: Association for Computing Machinery, 2019, p. 495504. [Online]. Available: https://doi.org/10.1145/3331184.3331241

[37] P. Schulam and I. Muslea, "Improving the exploration/exploitation trade-off in web content discovery."

[38] A. Kolobov, Y. Peres, C. Lu, and E. J. Horvitz, "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., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/ad13a2a07ca4b7642959dc0c4c740ab6-Paper.pdf

[39] J. Cho and H. Garcia-Molina, "Parallel crawlers," in *Proceedings of the 11th International Conference on World Wide Web*, 2002, p. 124135.

[40] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Software: Practice and Experience*, vol. 34, no. 8, pp. 711–726, 2004.

[41] D. H. Chau, S. Pandit, S. Wang, and C. Faloutsos, "Parallel crawling for online social networks," in *Proceedings of the 16th International Conference on World Wide Web*, 2007, p. 12831284.

[42] M. Bošnjak, E. Oliveira, J. Martins, E. 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*, 2012, p. 12331240.

[43] Y. Guo, K. Li, K. Zhang, and G. 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)*, 2006, pp. 745–748.

[44] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang, "iRobot: an intelligent crawler for web forums," in *Proceedings of the 17th International Conference on World Wide Web*, 2008, p. 447456.

[45] I. Hernández, C. R. Rivero, and D. Ruiz, "Deep Web crawling: a survey," *World Wide Web*, vol. 22, pp. 1577–1610, 2019.

[46] V. Crescenzi, P. Merialdo, and P. Missier, "Fine-grain web site structure discovery," in *Proc. WIDM*, 2003, p. 1522.

[47] ——, "Clustering web pages based on their structure," *Data and Knowledge Engineering*, vol. 54, no. 3, pp. 279–299, 2005.

[48] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, pp. 285–294, 1933.

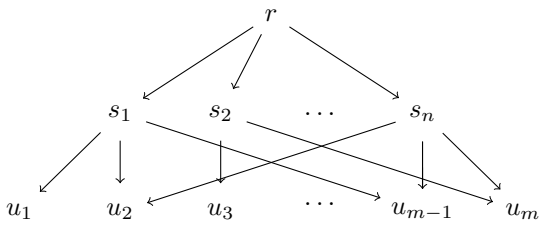


Fig. 5: Graphical summarization of the graph G_{sc}

APPENDIX

SUPPLEMENTARY MATERIAL FOR SECTION II (PROBLEM STATEMENT AND OUR MODELING)

Proposition 4. *Given a website graph $G = (V, E, r, \omega, \lambda)$, a subset $V^* \subseteq V$ and some $k \in \mathbb{R}^+$, determining whether there exists a crawl $T = (V', E')$ of G such that $V^* \subseteq V'$ and $\omega(T) \leqslant k$ is NP-complete; hardness holds even when ω and λ are constant functions.*

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 $\omega(T) \leqslant k$. 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 k, determining whether there exists a *cover* $\mathcal{C} \subseteq \{s_1, \ldots, s_n\}$ such that $|\mathcal{C}| \leqslant k$ and:

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

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

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, k has representation size $\Theta(\log k)$. This yields a total input size of $\Theta((\log m)\,(\sum_{i=1}^{n} |s_i| + 1) + \log k)$. Note that $\sum_{i=1}^{n} |s_i| \geqslant \max(m, n)$ so this is $\Omega(\max(m, n) \log m + \log k)$. 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 $\mathcal{C} \subseteq \{s_1, \ldots, s_n\}$ such that $|\mathcal{C}| \leqslant k$ and $\bigcup_{i \in \mathcal{C}} s_i = \mathcal{U}$. Then consider the crawl T_{sc} of G_{sc} formed by including r, every element of $\mathcal{C}$ using the edge from r to that element, and every edge from an element of $\mathcal{C}$ to an element of $\mathcal{U}$. Since $\mathcal{C}$ is a cover, this includes all elements of $\mathcal{U}$. The total cost of this crawl $\omega(T_{\text{sc}}) = 1 + |\mathcal{C}| + |\mathcal{U}| \leqslant |\mathcal{U}| + k + 1$.

Now suppose that there exists a crawl T_{sc} of G_{sc} of total cost $\omega(T_{\mathrm{sc}}) \leqslant |\mathcal{U}| + k + 1$. 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 $\leqslant k$ vertices are therefore vertices of $\mathcal{S}$. We pose $\mathcal{C}$ to be those. Then $|\mathcal{C}| \leqslant k$ and since T_{sc} is a crawl, for every $u \in \mathcal{U}$, there exists at least one $s \in \mathcal{C}$ such that the edge (s, u) is in T_{sc}, meaning that $u \in s$. We indeed have $\mathcal{U} = \bigcup_{s \in \mathcal{C}} s$. □

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
```

SUPPLEMENTARY MATERIAL FOR SECTION IV (EXPERIMENTAL RESULTS)

This section presents detailed experimental results that could not fit in the paper. Figures 6 to 11 present exhaustive crawler and baselines performance of the 18 websites presented in Table I. 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. Tables VII to XVII 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 VII: 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 VIII: 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 IX: 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 X: 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 XI: 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 XII: 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 XIII: 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

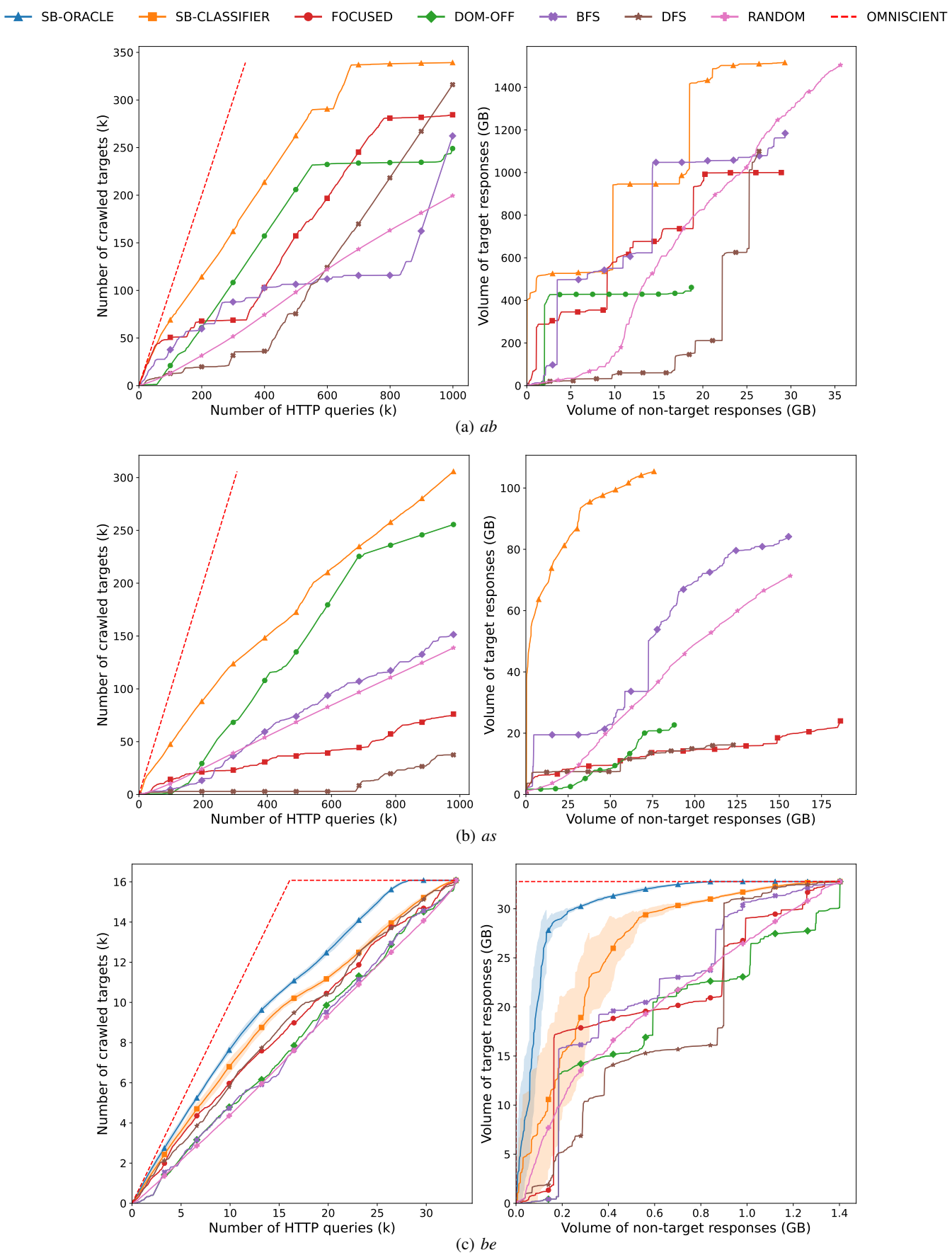
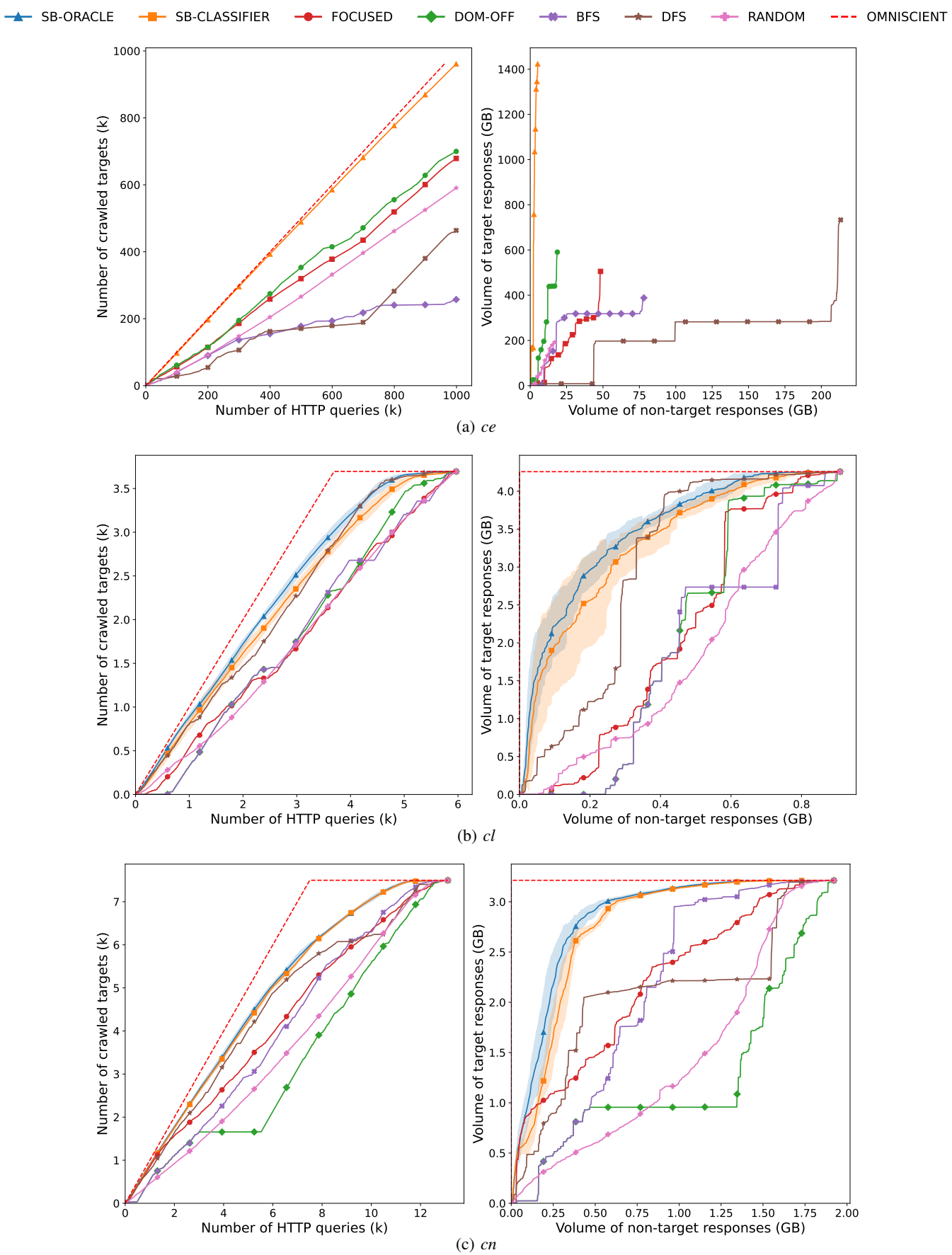


Fig. 6: Comparison of different crawler performance for websites *ab*, *as*, and *be*

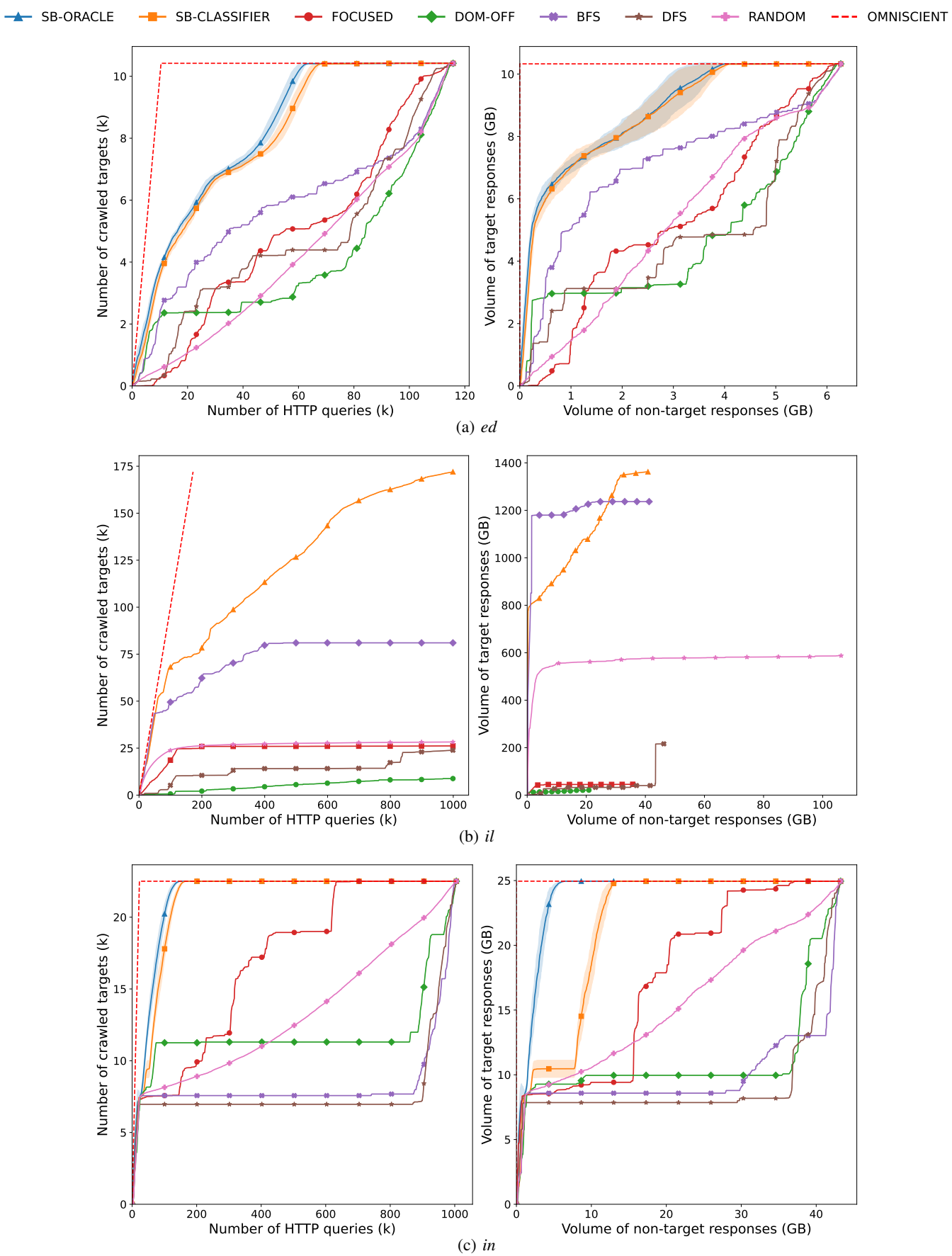


(a) *ce*

(b) *cl*

(c) *cn*

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



(a) *ed*

(b) *il*

(c) *in*

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

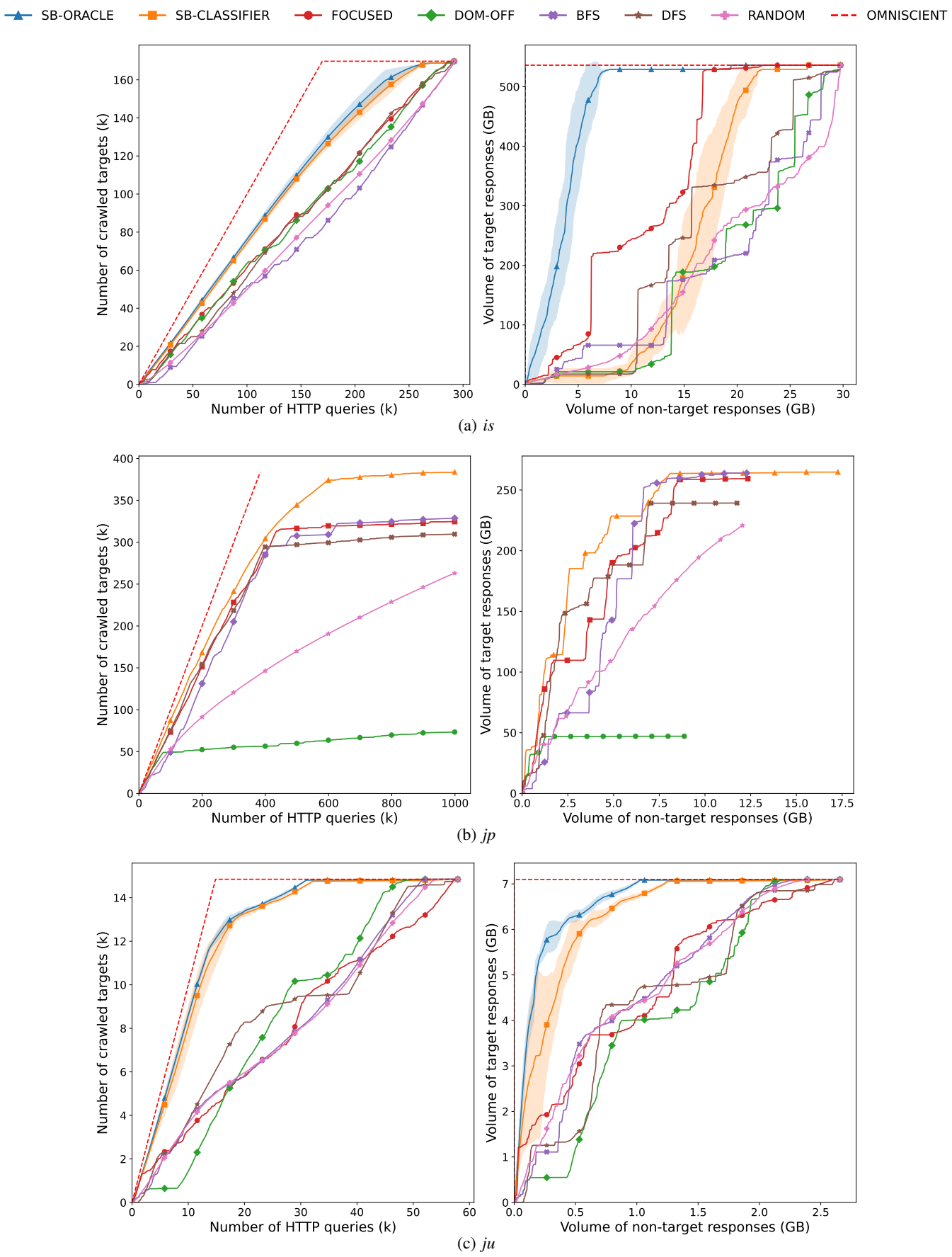


Fig. 9: Comparison of different crawler performance for websites *is*, *jp*, and *ju*

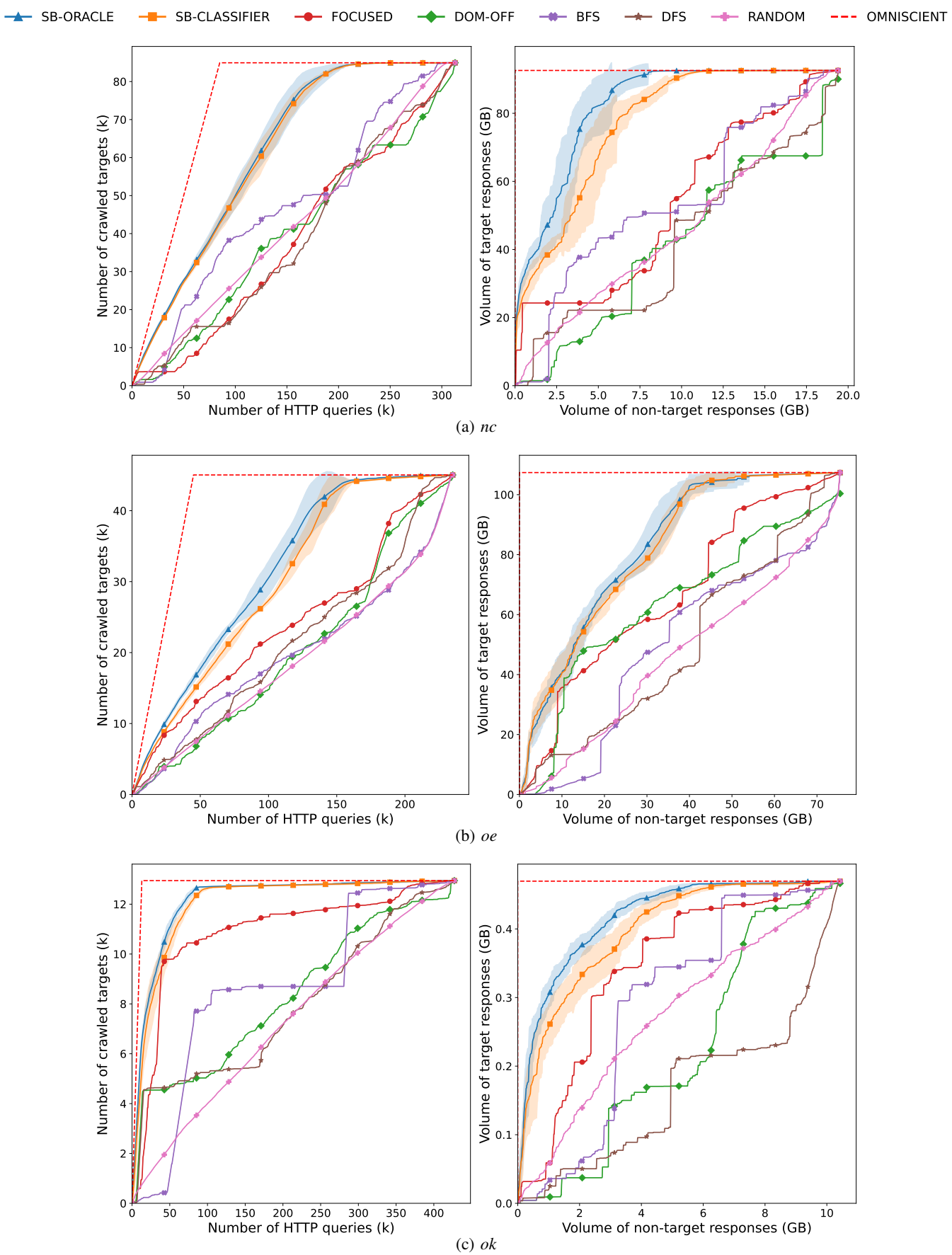
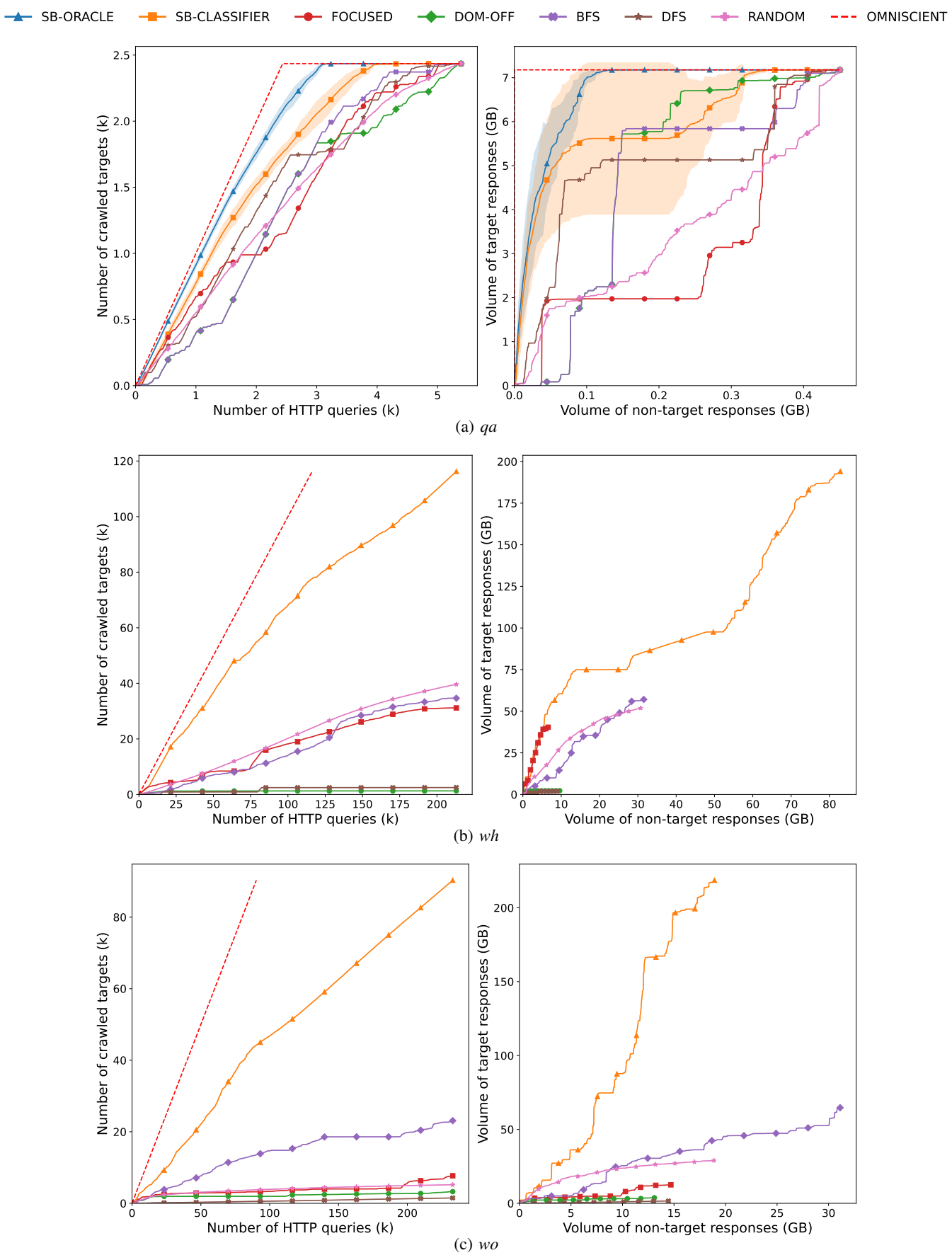


Fig. 10: Comparison of different crawler performance for websites *nc*, *oe*, and *ok*

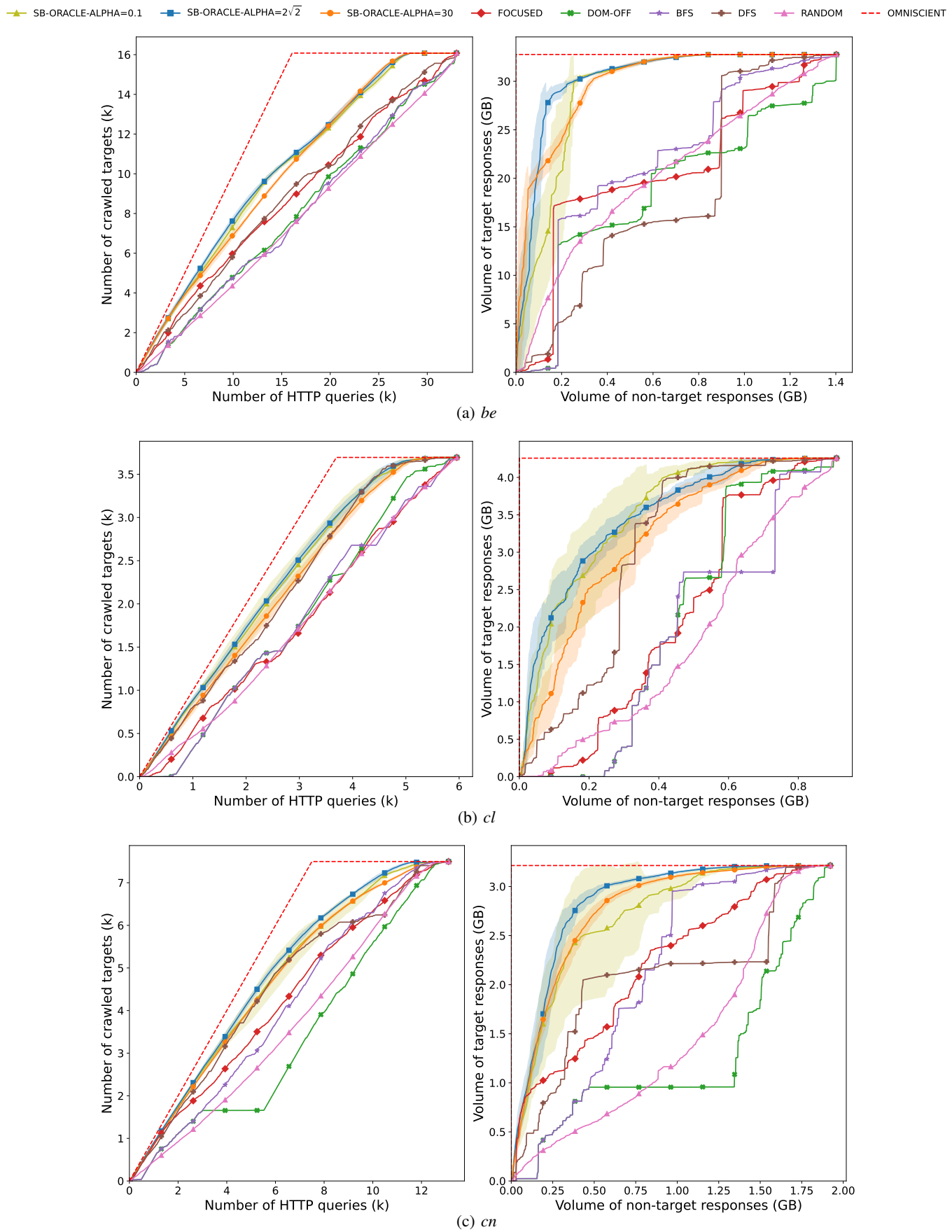


(a) *qa*

(b) *wh*

(c) *wo*

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

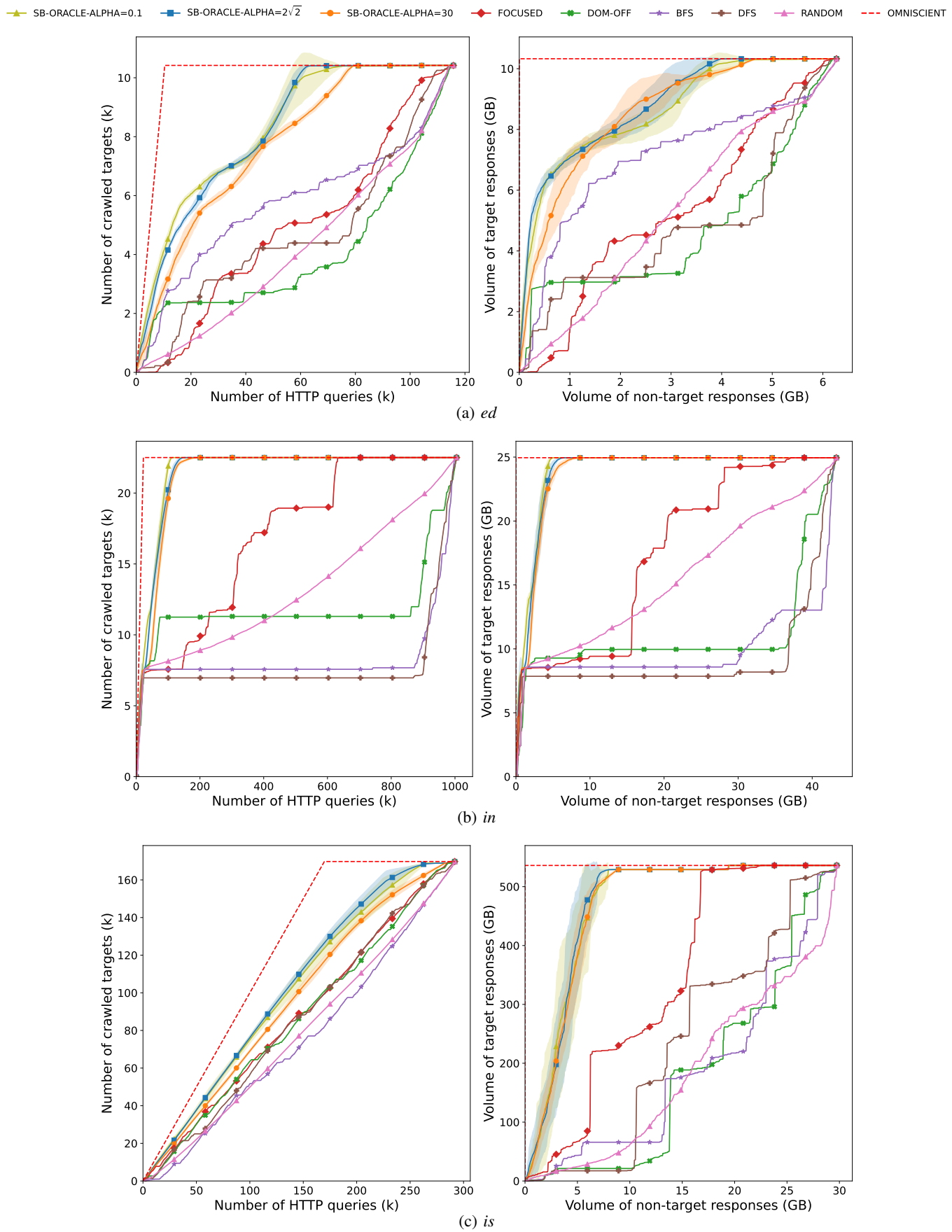


(a) *be*

(b) *cl*

(c) *cn*

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

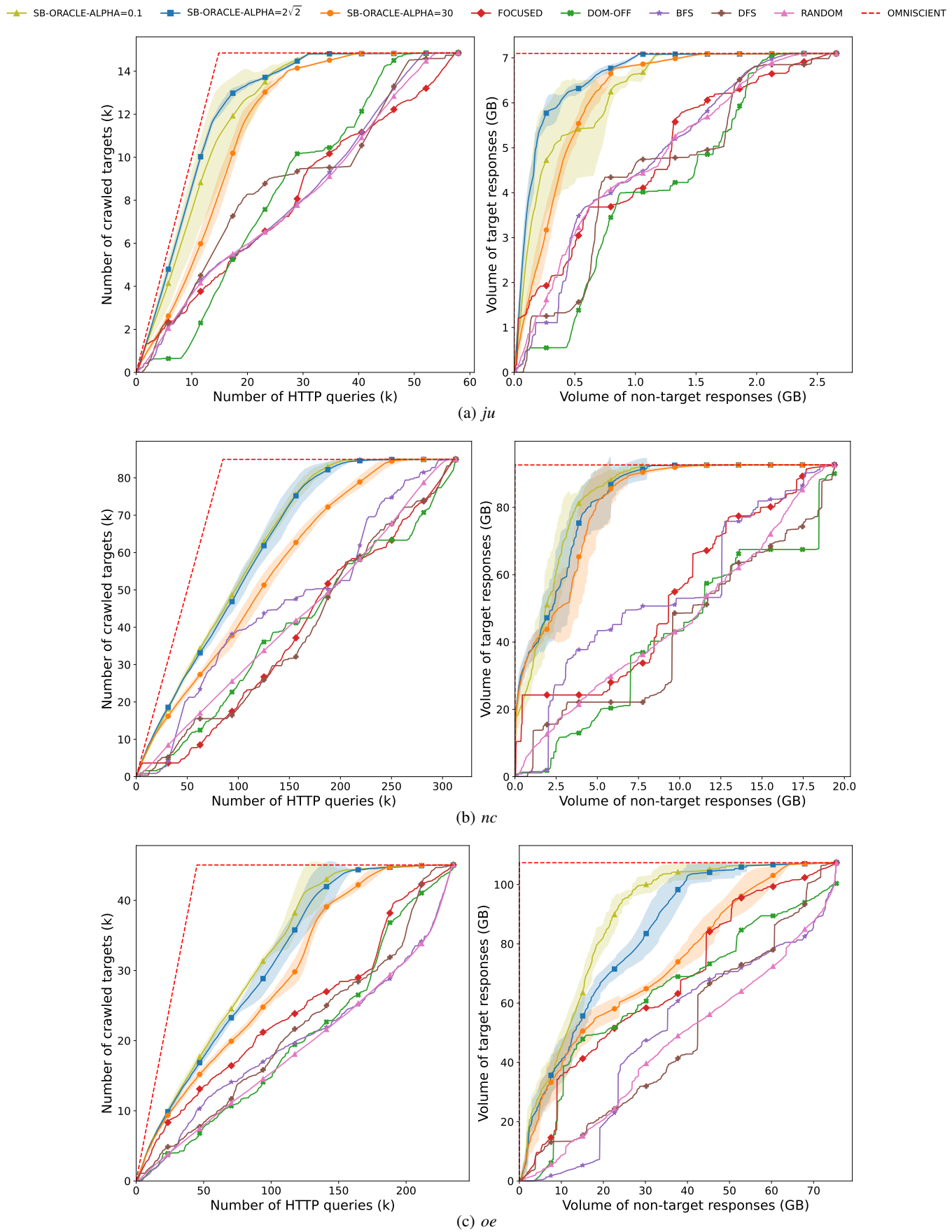


(a) *ed*

(b) *in*

(c) *is*

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

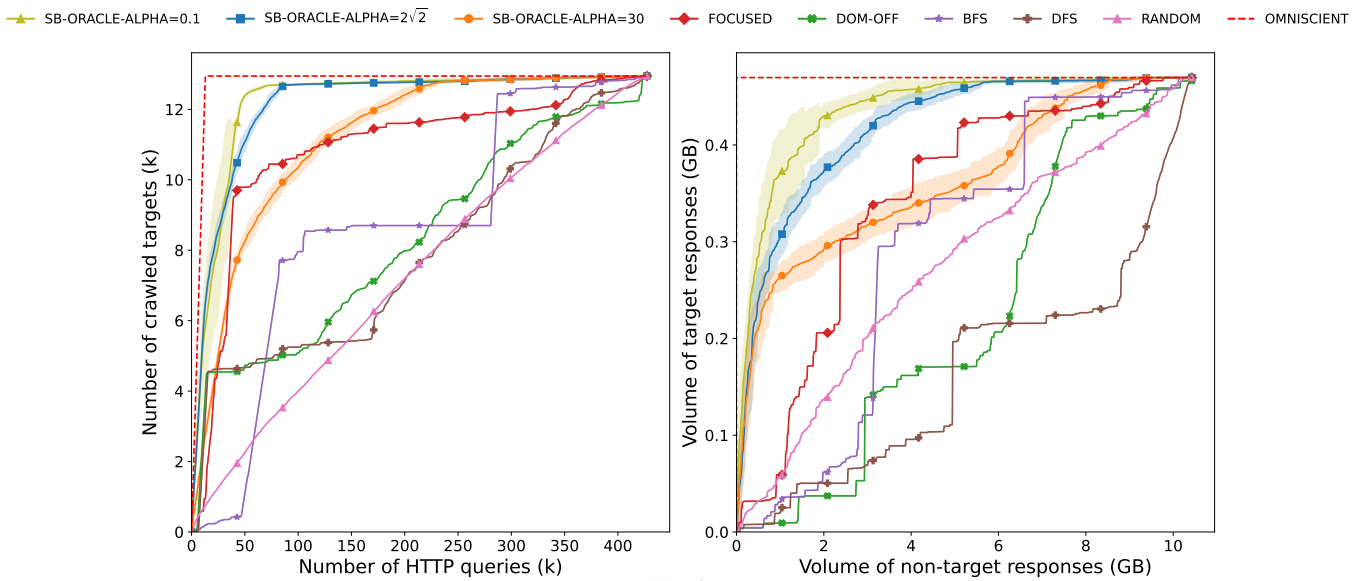


(a) *ju*

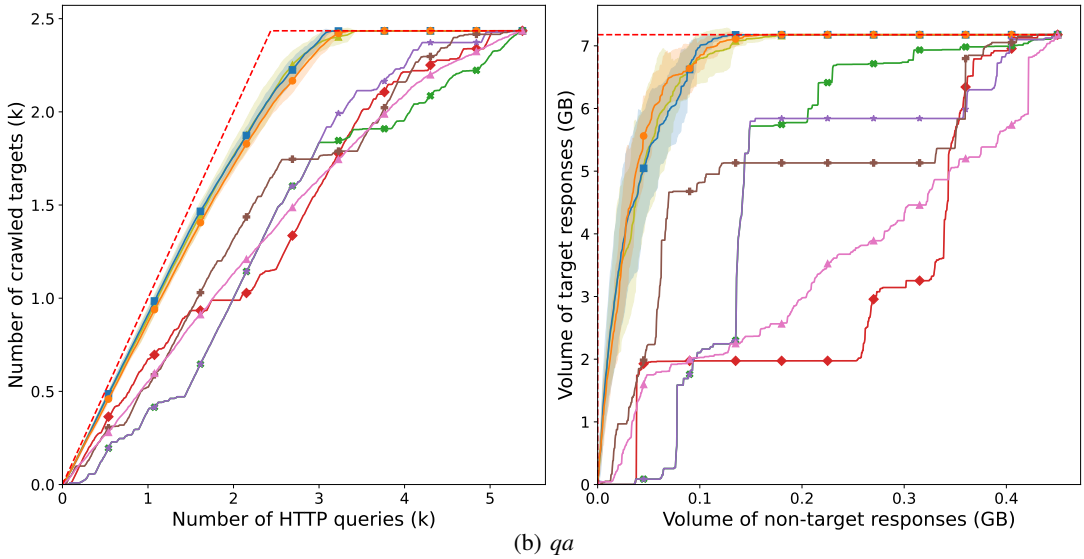
(b) *nc*

(c) *oe*

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



(a) *ok*



(b) *qa*

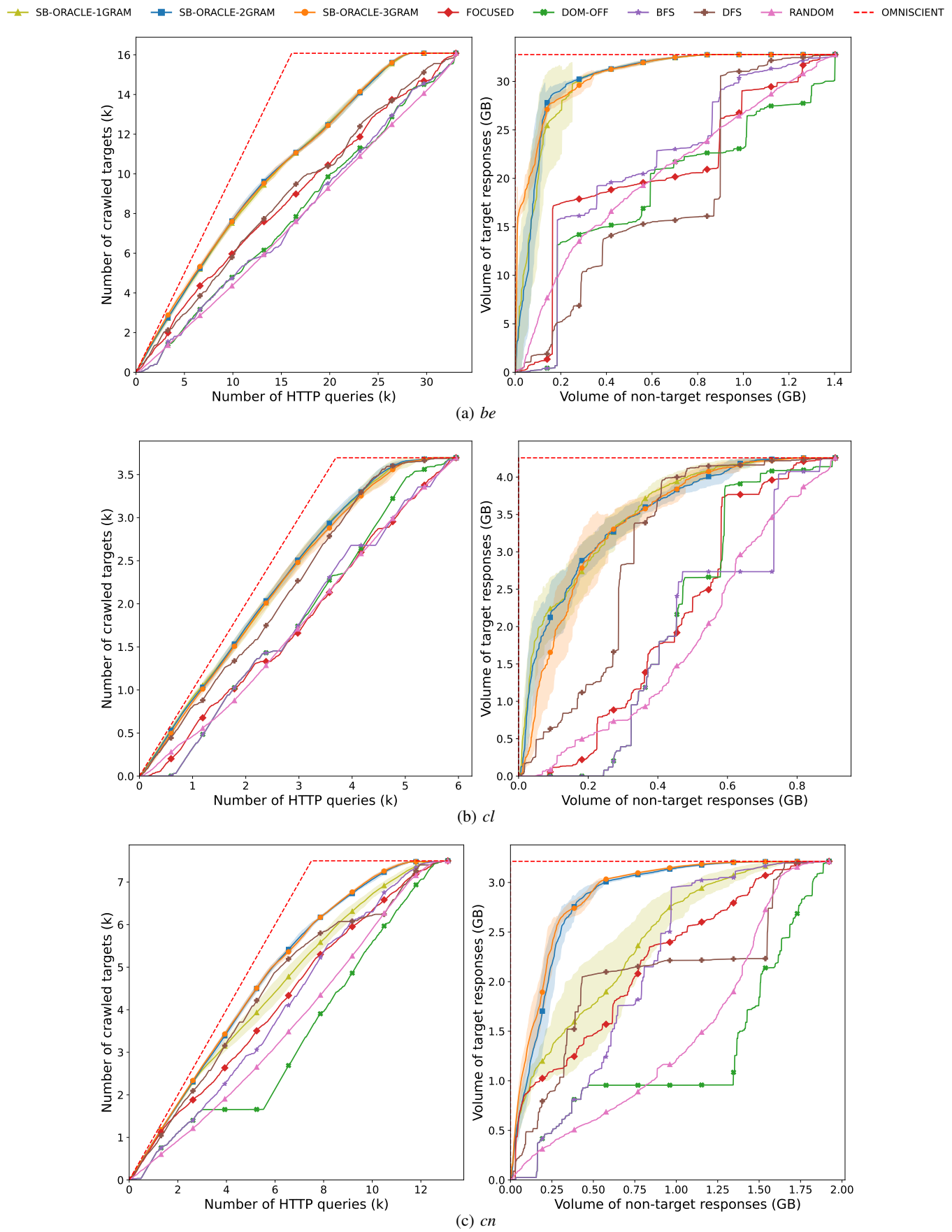
Fig. 15: Crawler performance for hyper-parameter study on exploration–exploitation coefficent α, for websites *ok*, and *qa*

TABLE XIV: 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 XV: 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

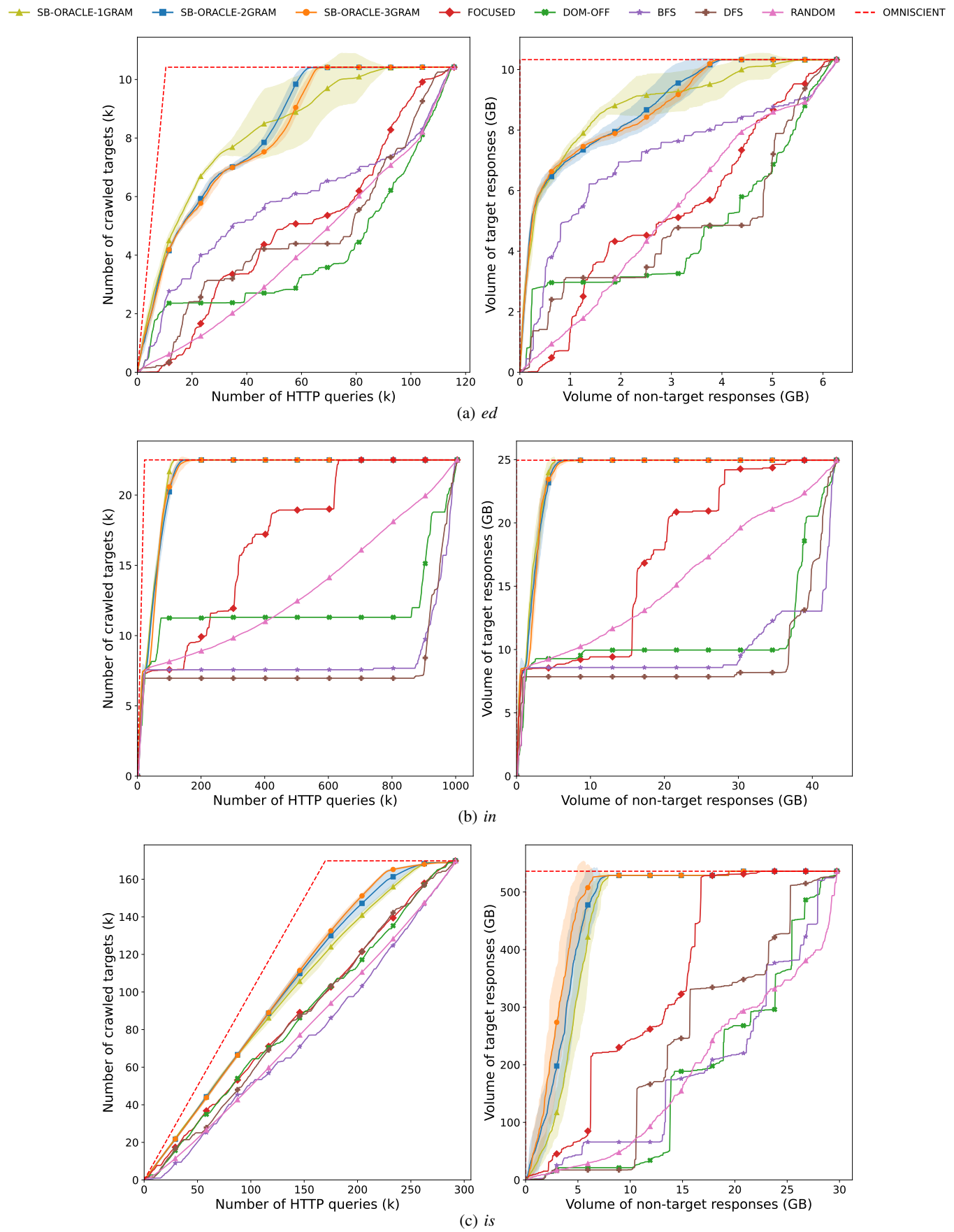


(a) *be*

(b) *cl*

(c) *cn*

Fig. 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*

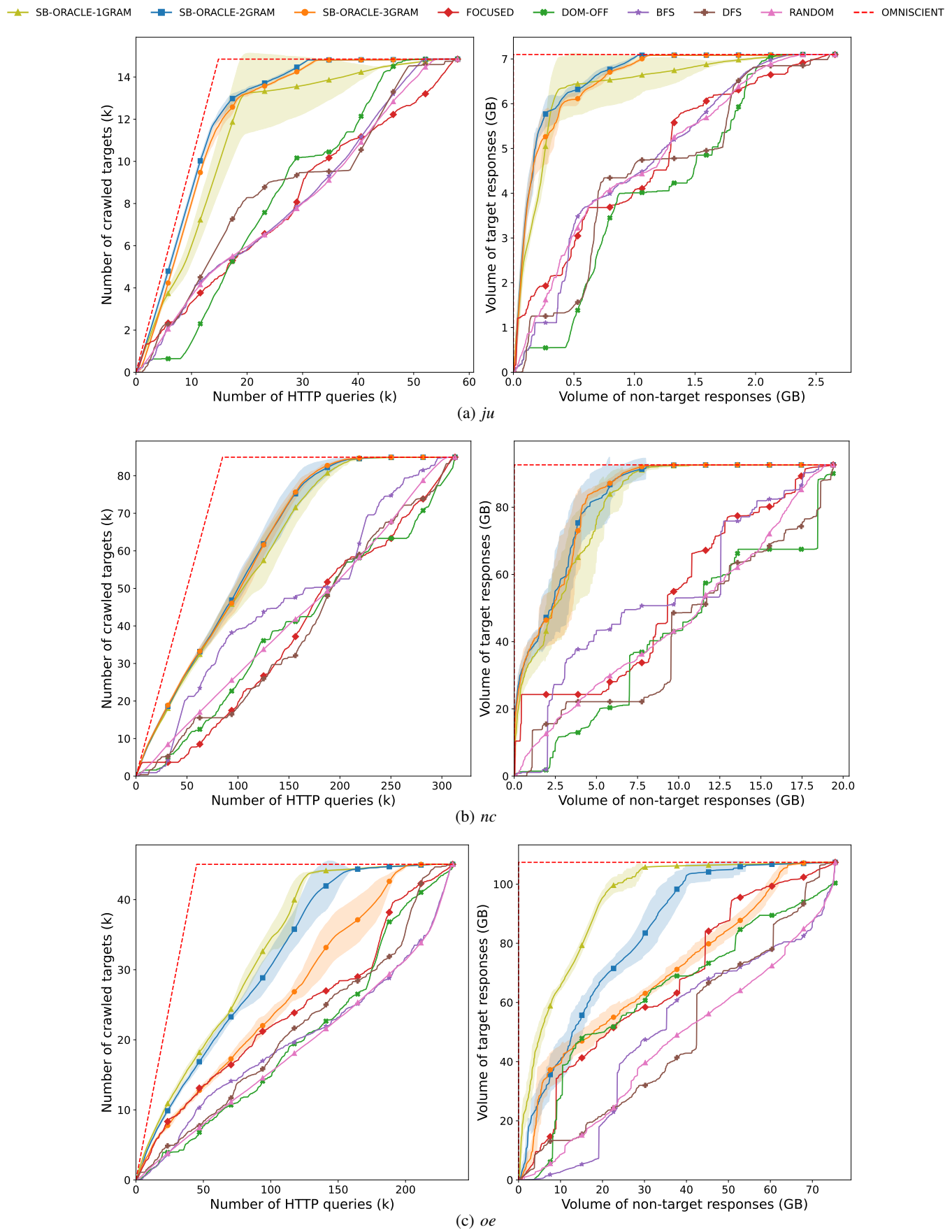


(a) *ed*

(b) *in*

(c) *is*

Fig. 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*

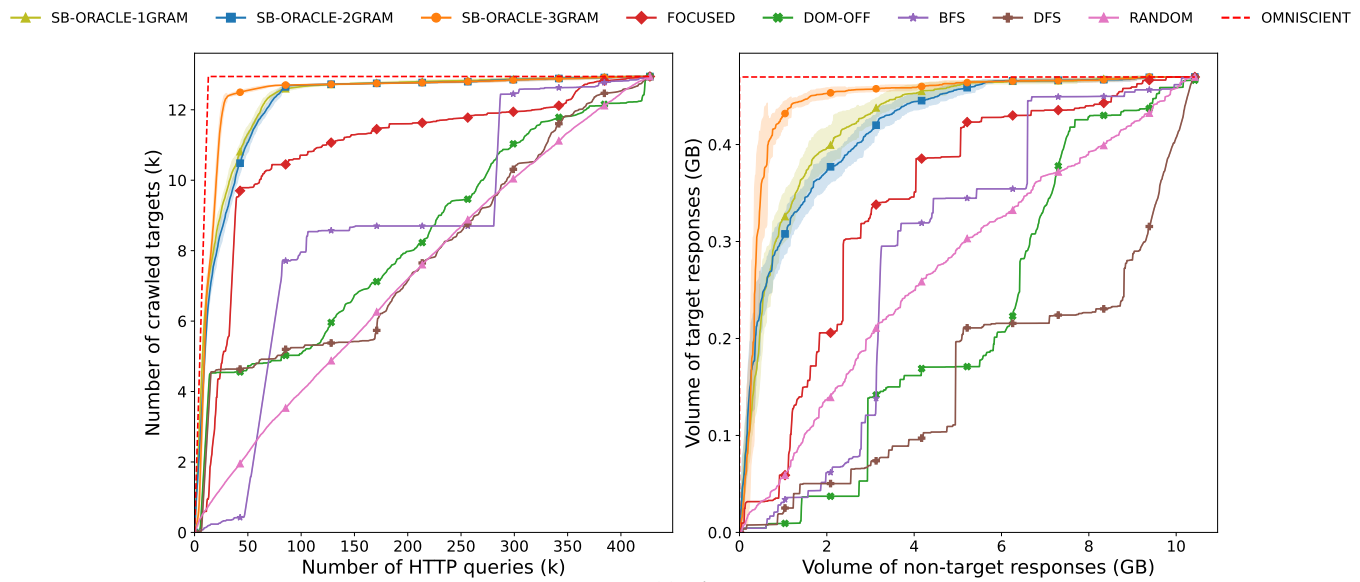


(a) *ju*

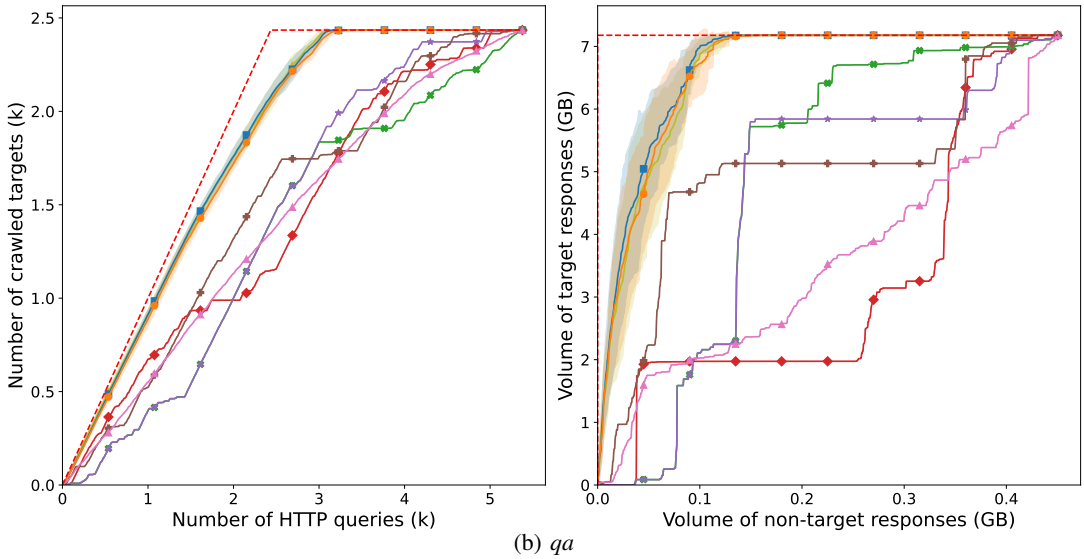
(b) *nc*

(c) *oe*

Fig. 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*

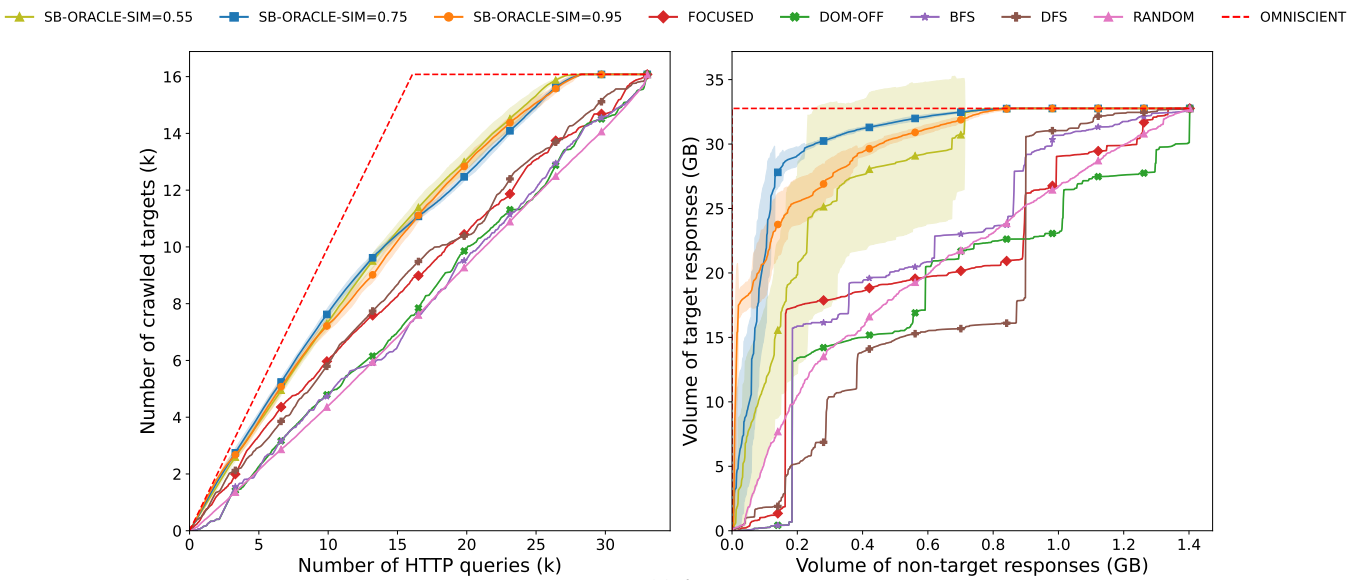
Fig. 19: Crawler performance for impact study of the choice of n in n-grams used in DOM path vector representation, for websites *ok*, and *qa*

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

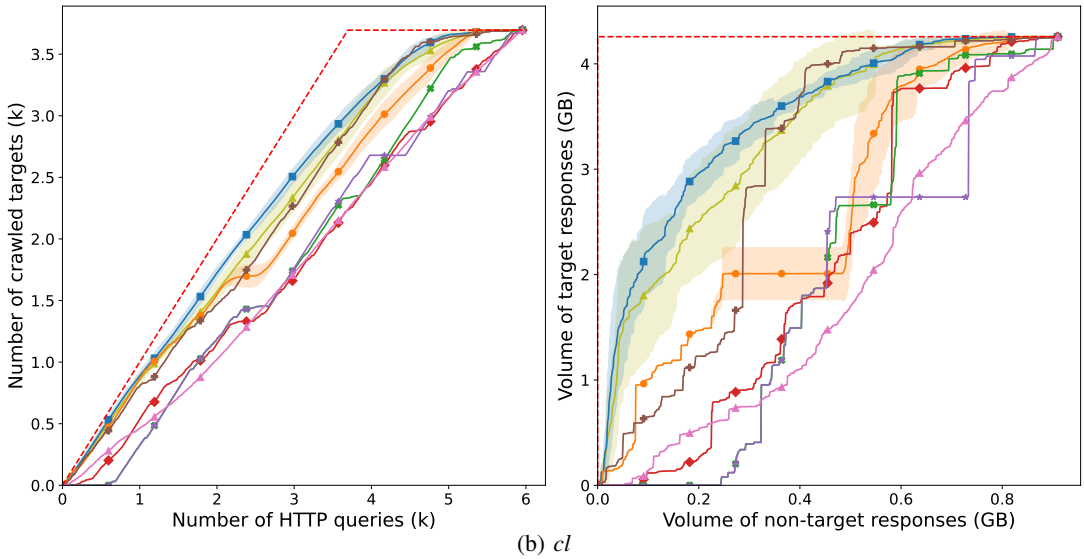
True/Predicted	HTML	Target	Neither
HTML	391802	3556	0
Target	669	12058	0
Neither	12484	2206	0

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

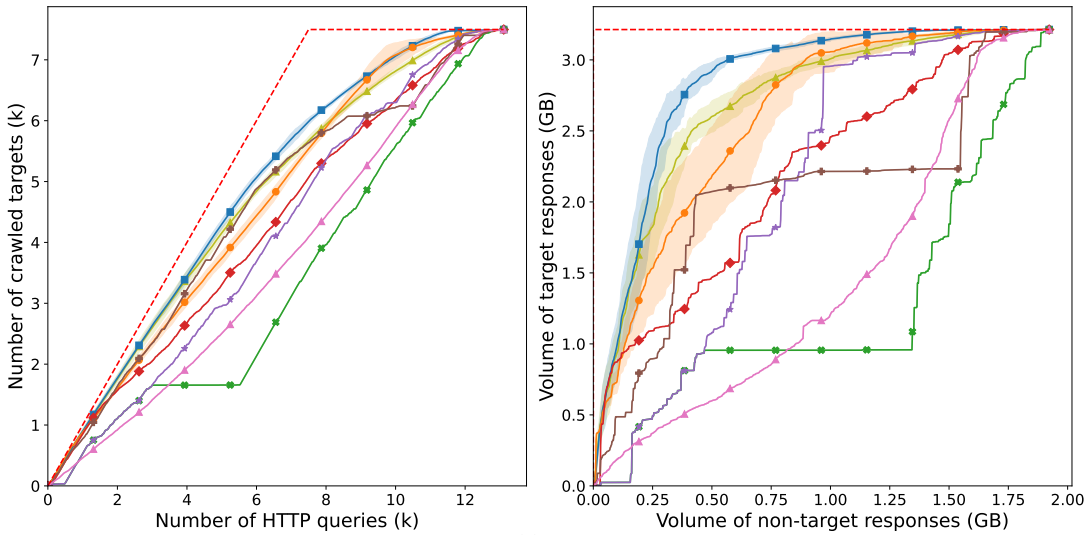
True/Predicted	HTML	Target	Neither
HTML	1311	31	0
Target	77	2356	0
Neither	1320	247	0



(a) *be*

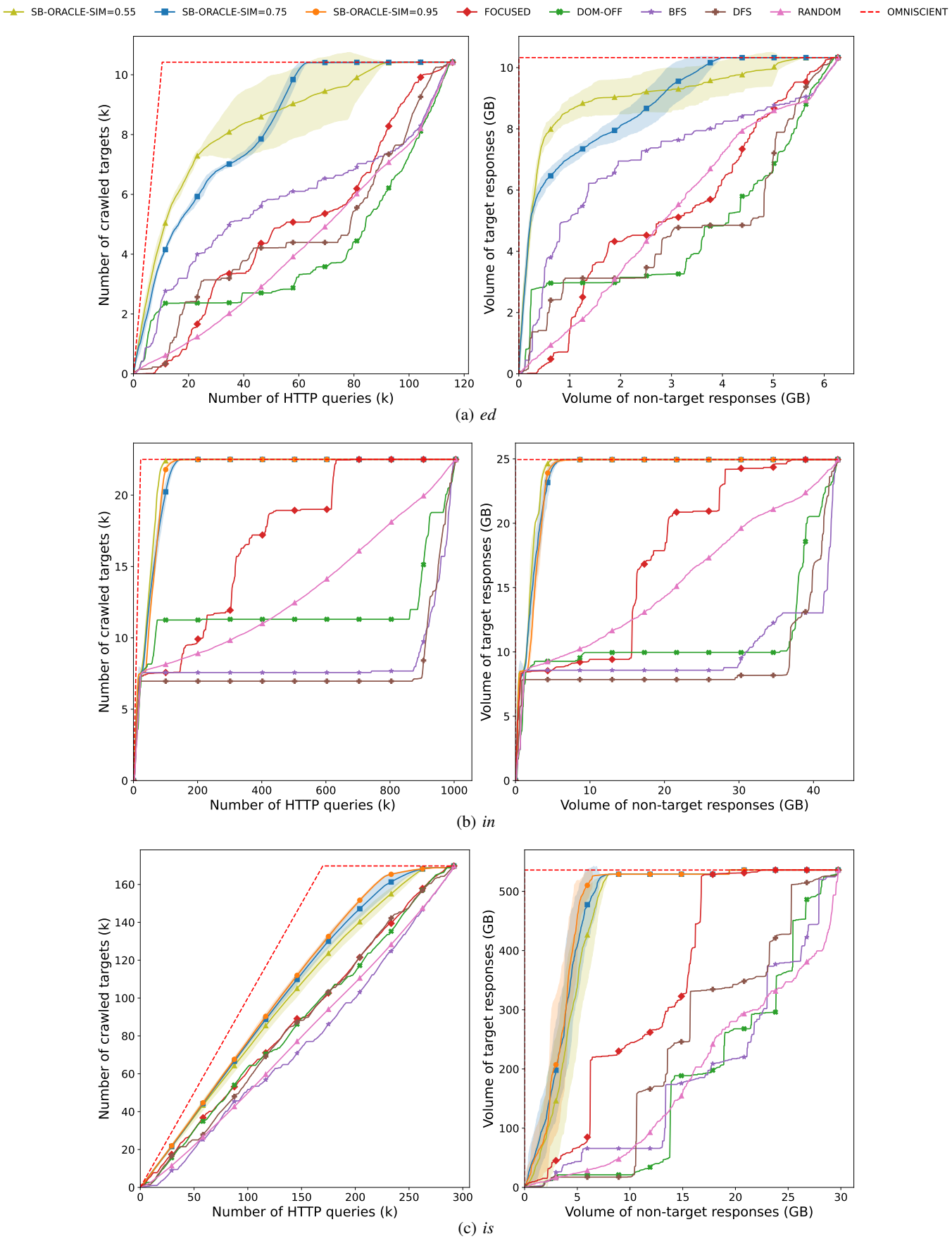


(b) *cl*



(c) *cn*

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

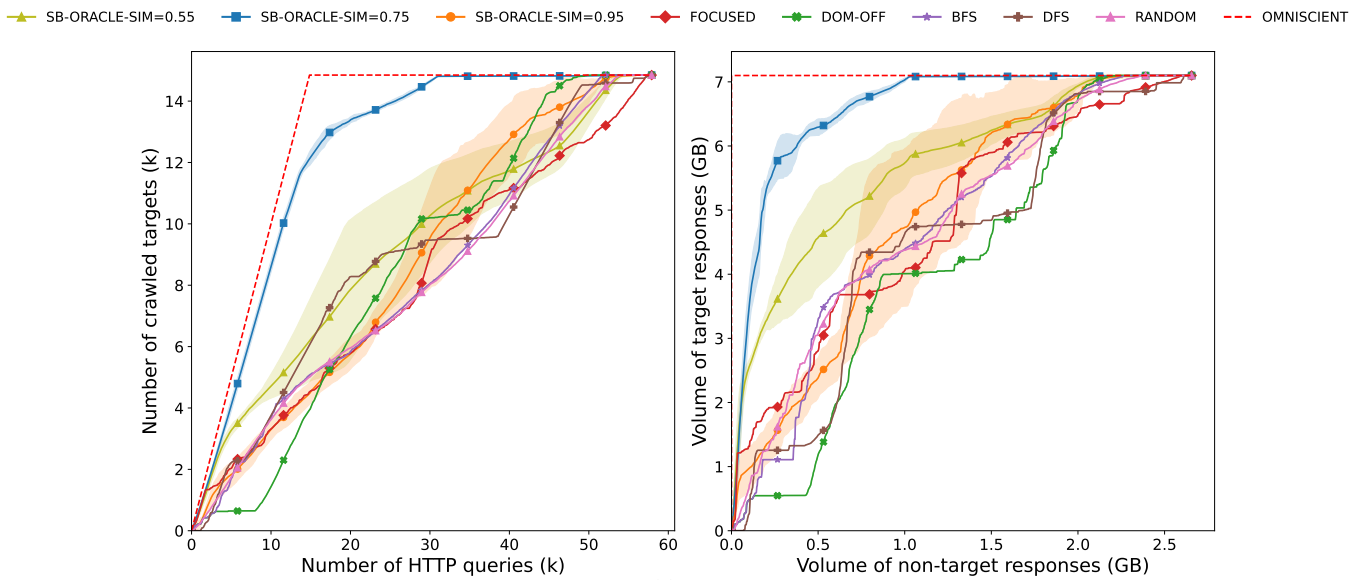


(a) *ed*

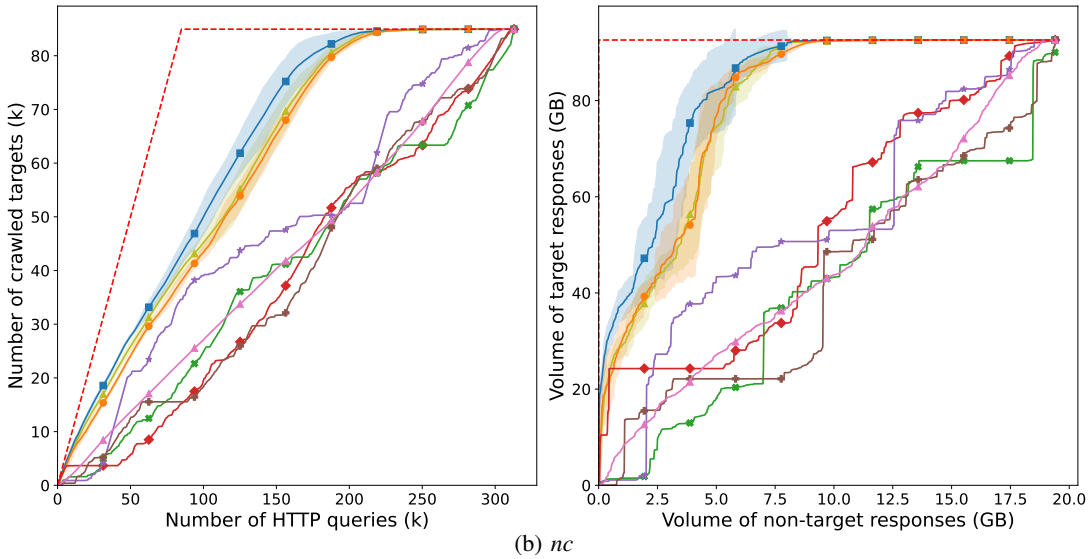
(b) *in*

(c) *is*

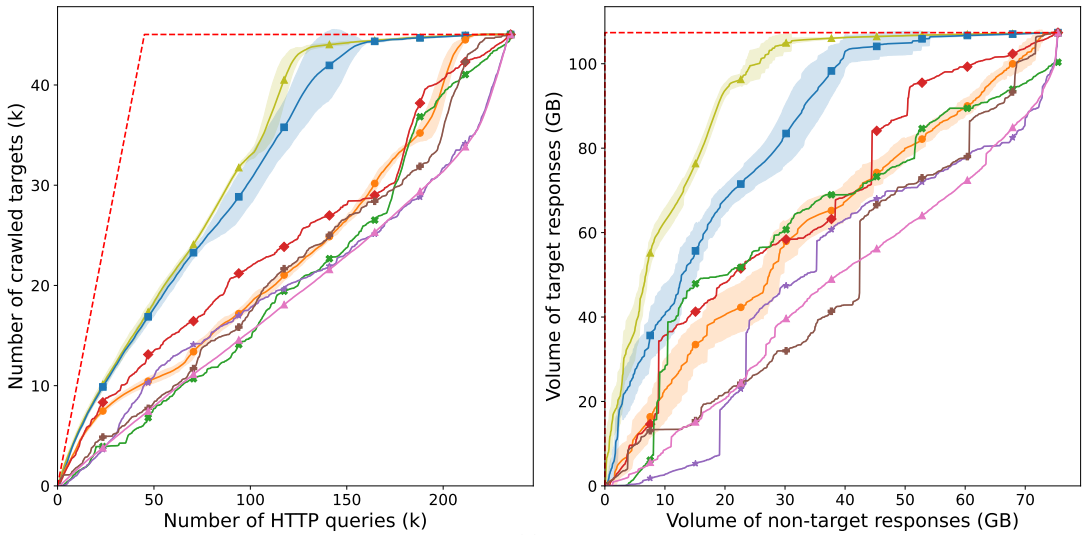
Fig. 21: Crawler performance for impact study on similarity threshold θ, for websites *ed*, *in*, and *is*



(a) *ju*

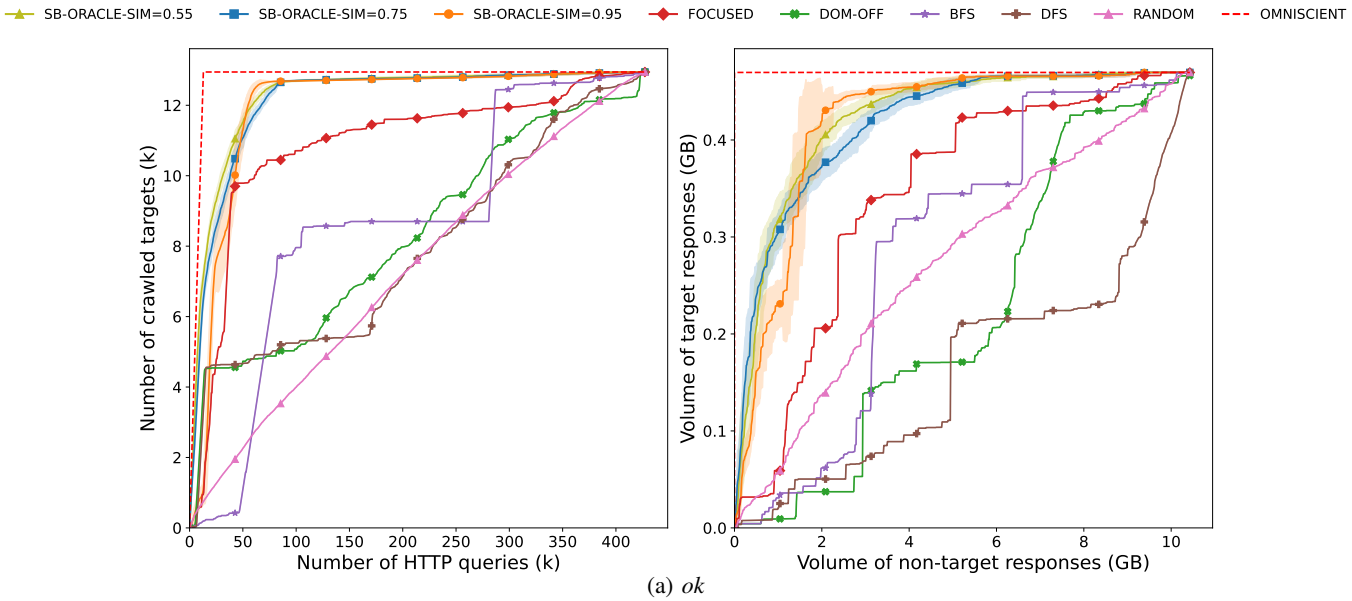


(b) *nc*

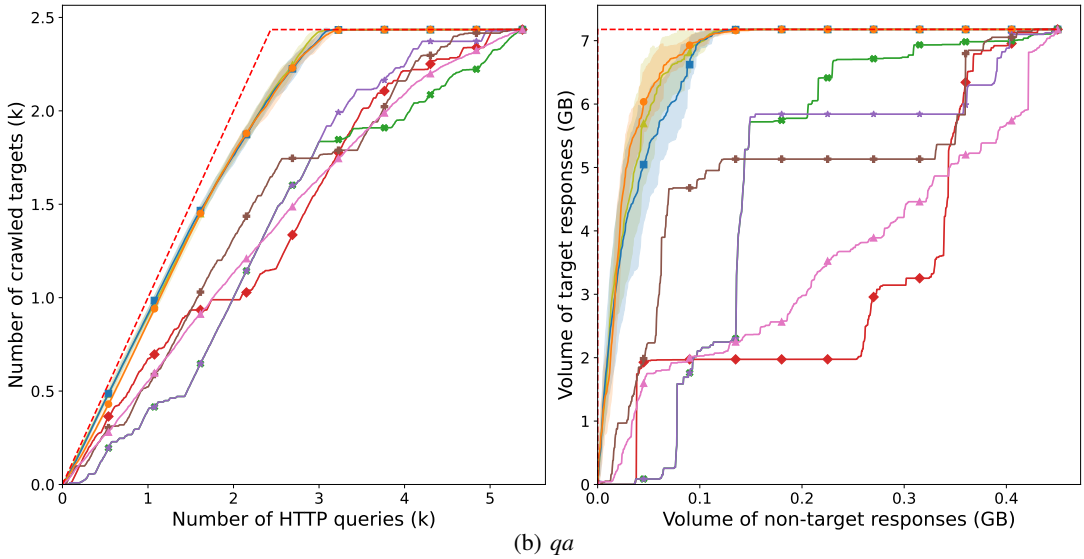


(c) *oe*

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



(a) *ok*



(b) *qa*

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

SUPPLEMENTARY MATERIAL FOR SECTION V (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. II-A). 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. VI). 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. IV-A), 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[7] 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.

[7]https://archive.org/

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 12331240, 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.

[CGM02] Junghoo Cho and Hector Garcia-Molina. Parallel crawlers. In *Proceedings of the 11th International Conference on World Wide Web*, page 124135, 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 12831284, 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 447456, 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 576587. 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 495504, 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 432440, 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.