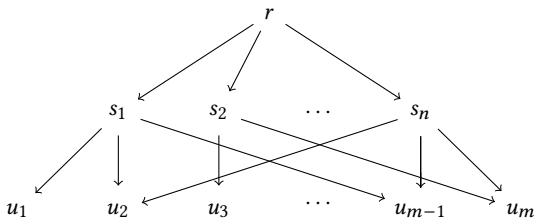


Figure 4: Graphical summarization of the graph G_{sc}

A SUPPLEMENTARY MATERIAL FOR SECTION 2 (GRAPH CRAWLING PROBLEM)

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* $C \subseteq \{s_1, \ldots, s_n\}$ such that $|C| \leqslant k$ and:

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

We now propose a polynomial-time many-one reduction of the set cover problem to an instance of the graph crawling problem. We create a website graph $G_{\text{sc}} = (V_{\text{sc}}, E_{\text{sc}}, r, \omega, \lambda)$ as follows. We set V_{sc} to be $\{u_1, \ldots, u_m, r, s_1, \ldots, s_n\}$, including representations for every element of the universe $\mathcal{U}$, every set of the collection $\mathcal{S}$, as well as a distinct root r (by abuse of notation, we do not distinguish between elements of $\mathcal{U}$, $\mathcal{S}$ and the way they are represented in V_{sc}). We define E_{sc} as $\{(r, s_i) \mid i \in \{1, \ldots, n\}\} \cup \{(s_i, u) \mid u \in s_i, i \in \{1, \ldots, n\}\}$. In other words, in G_{sc} from the origin (root) r, we model each element of $\mathcal{S}$ as a vertex, that can be reached following a dedicated (directed) edge. Finally, for each new vertex $s_i \in \mathcal{S}$, we have as many outgoing edges as there are elements of $\mathcal{U}$ in s_i. Finally, we set ω to be the constant function that assigns cost 1 to every vertex, and λ to be some constant function. The result is a graph in the form of a tree of depth 2, depicted in Figure 4. We fix V^* to be $\mathcal{U}$. We state that there exists $C \subseteq \{s_1, \ldots, s_n\}$ such that $|C| \leqslant k$ and $\bigcup_{i \in C} s_i = \mathcal{U}$ if and only if there exists a crawl T_{sc} of G_{sc} containing all elements of V^* and of total cost $\omega(T_{\text{sc}}) \leqslant |\mathcal{U}| + k + 1$.

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 $\Theta((\log m)\left(\sum_{i=1}^{n} |s_i| + 1\right) + \log k)$. 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 4 can clearly be done in time polynomial in m and n (namely, in $O(m \times n)$ in the worst case where every set of the collection contains every element). The reduction is therefore polynomial-time.

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

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

□

B SUPPLEMENTARY MATERIAL FOR SECTION 3 (DATA ACQUISITION AS GRAPH CRAWLING)

Here is the full list of the 37 MIME types used to identify targets in our implementation:

```
application/csv
application/json
application/msword
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
```

C SUPPLEMENTARY MATERIAL FOR SECTION 4 (CRAWLING BASED ON REINFORCEMENT LEARNING)

We provide additional detailed algorithms for the different parts of our system, complementing the textual description in Section 4.

Algorithm 2 presents the process of mapping an action to a given hyperlink.

Algorithm 2: Finding the action for a hyperlink

Input: Action set A, mapped and projected hyperlink 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}

Algorithm 3 shows how we train and then exploit the URL classifier C.

Algorithm 3: Online URL classifier procedure

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

Algorithm 4 presents the high-level overview of the crawling procedure presented in this work.

Algorithm 4: 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**
    $a_{\text{c}} \leftarrow \arg\max_{a \in A} \mathbb{1}_a(t) \left(R_{\text{mean}}(a) + \alpha \sqrt{\frac{\log(t)}{N_t(a)+\varepsilon}} \right)$;
    $u \leftarrow$ 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 1);

D SUPPLEMENTARY MATERIAL FOR SECTION 5 (EXPERIMENTAL RESULTS)

This section presents detailed experimental results that could not fit in the paper. Figures 5, 6, and 7 present crawler performance on all websites regarding hyper-parameter studies on, respectively, exploration–exploitation coefficient α, n in the n-grams used in the DOM path vector representation, and similarity threshold θ. Tables 4 to 12 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 4: Confusion matrix of the URL classifier used in the SB algorithm on website *as* (on average, for 15 runs)

True/Predicted	HTML	Target	Neither
HTML	752074	2410	0
Target	464	66572	0
Neither	122237	2786	0

Table 5: 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 6: 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 7: Confusion matrix of the URL classifier used in the SB algorithm on website *ed* (on average, for 15 runs)

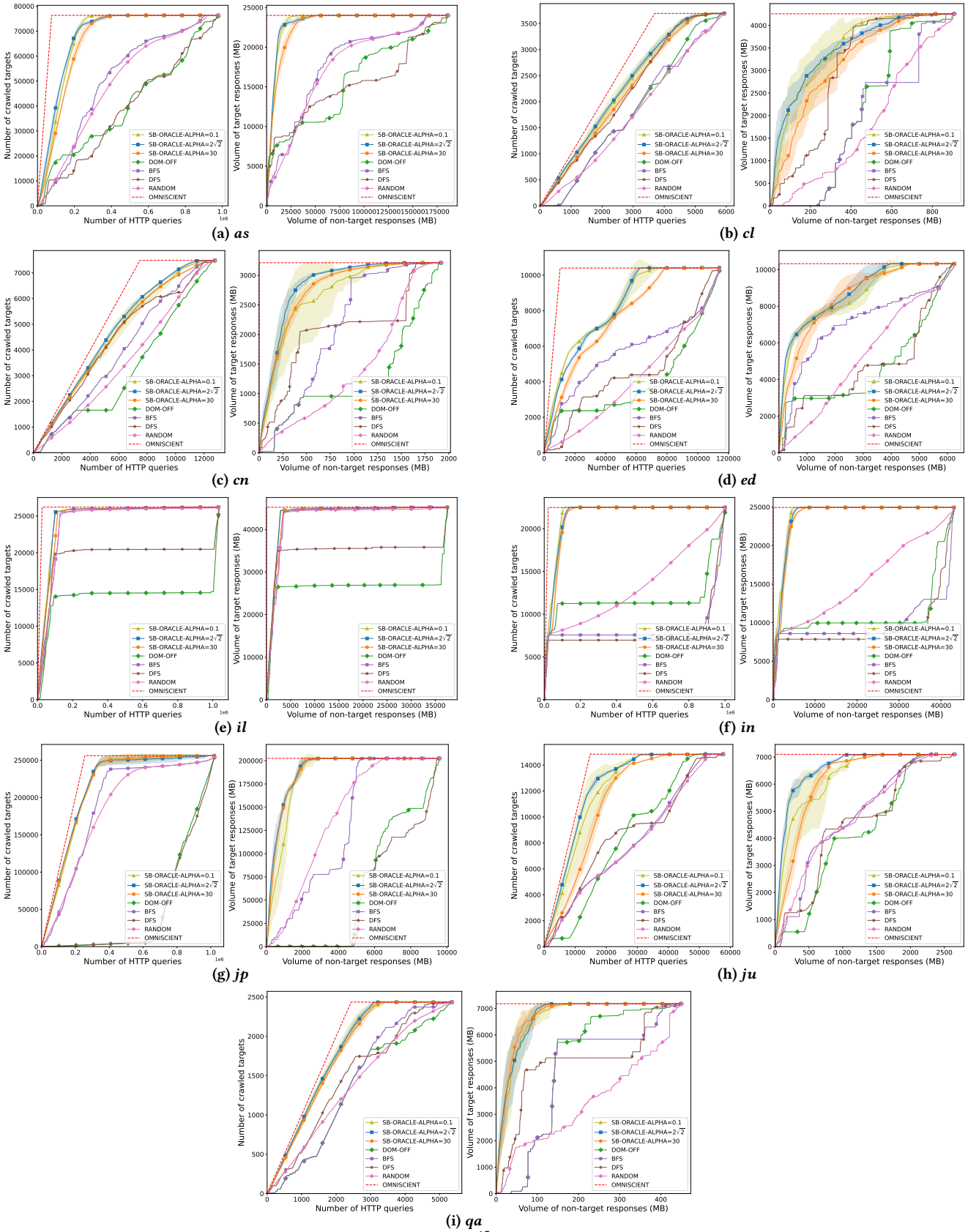
True/Predicted	HTML	Target	Neither
HTML	85920	2610	0
Target	175	10180	0
Neither	8645	1348	0

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

True/Predicted	HTML	Target	Neither
HTML	958618	1473	0
Target	61	25782	0
Neither	36003	2193	0

Table 9: 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



(a) *as* (b) *cl*

(c) *cn* (d) *ed*

(e) *il* (f) *in*

(g) *jp* (h) *ju*

(i) *qa*

Figure 5: Crawler performance for hyper-parameter study on exploration–exploitation coefficent α, for all websites

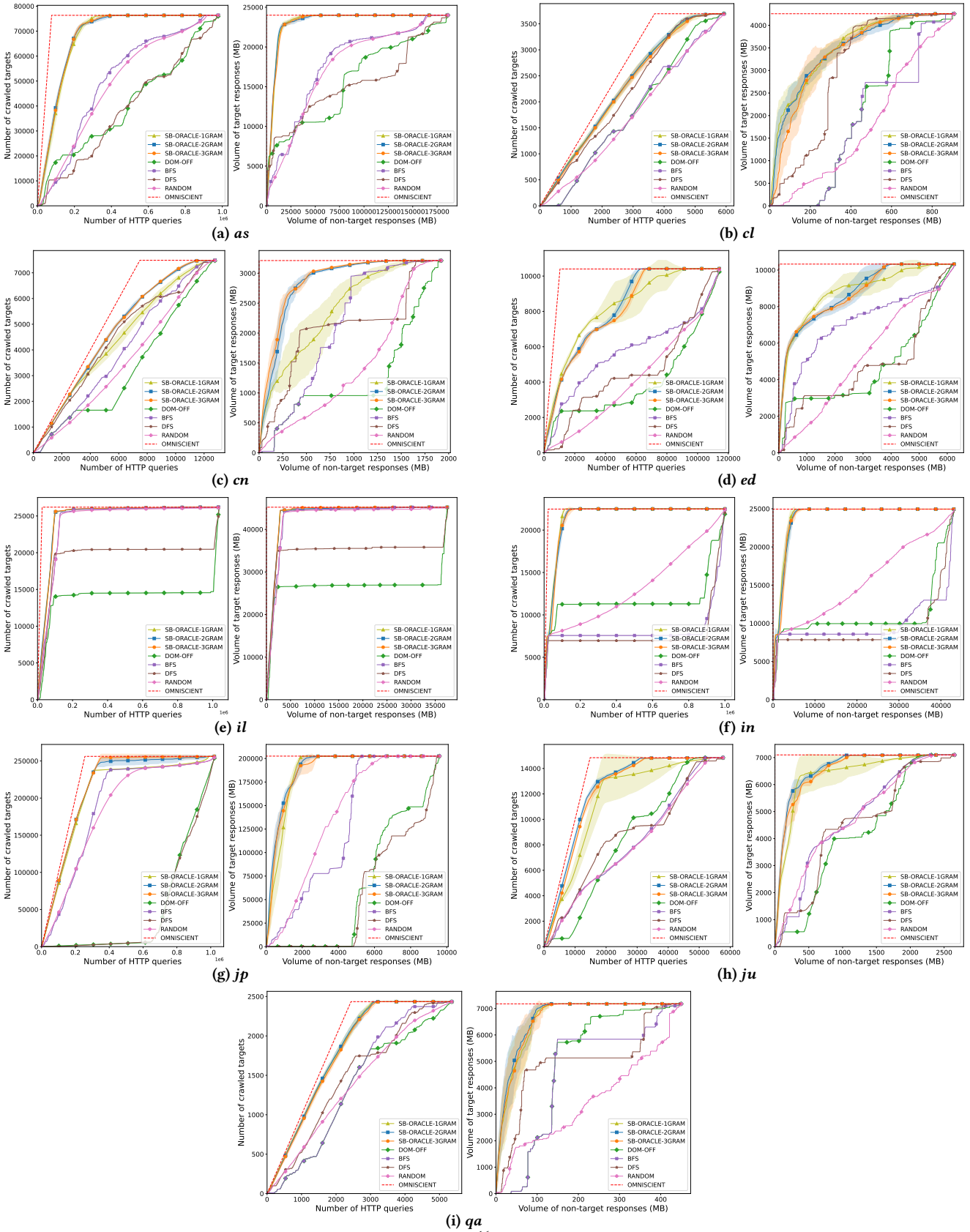
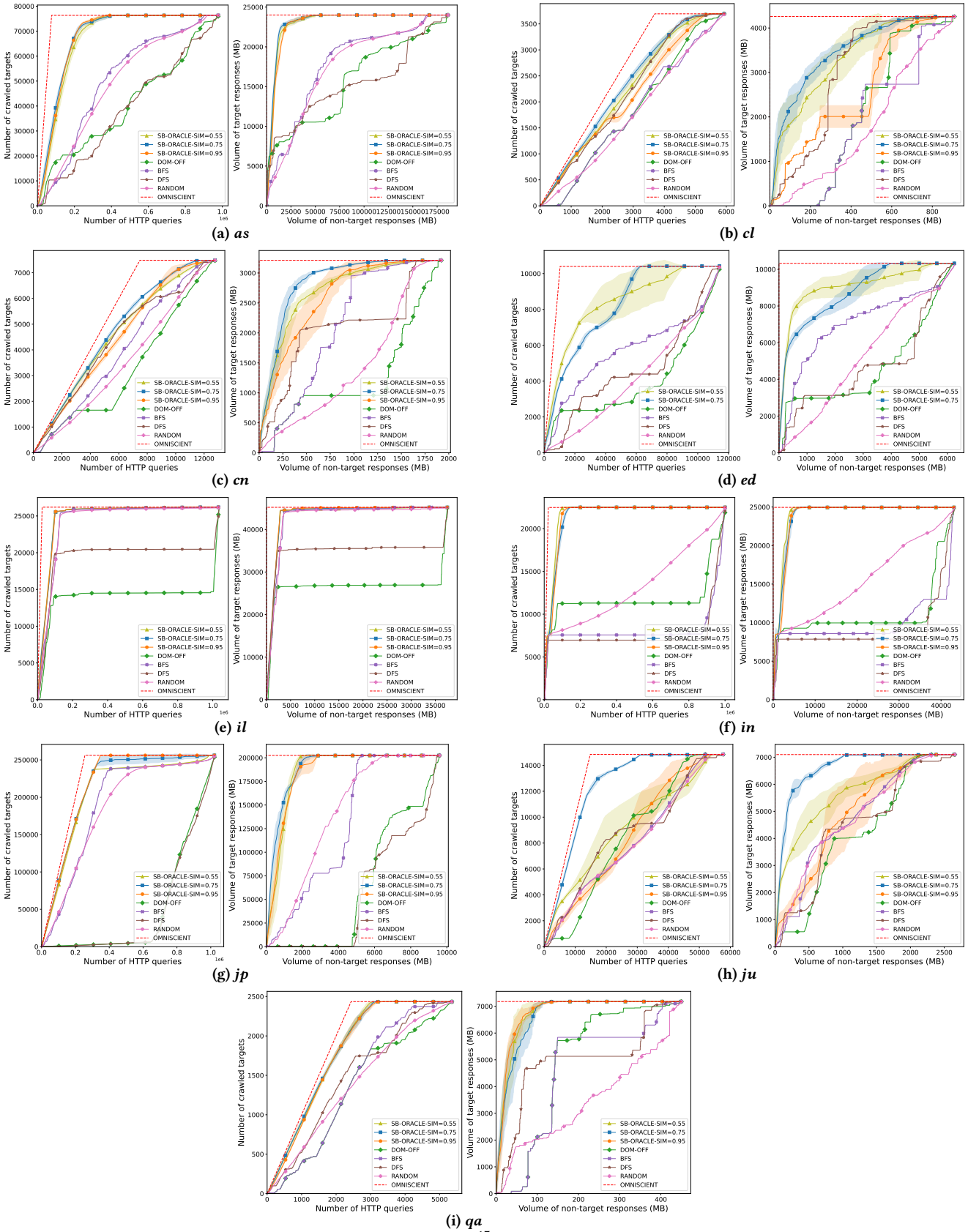


Figure 6: Crawler performance for impact study of the choice of n in n-grams used in DOM path vector representation, for all websites



(a) *as*

(b) *cl*

(c) *cn*

(d) *ed*

(e) *il*

(f) *in*

(g) *jp*

(h) *ju*

(i) *qa*

Figure 7: Crawler performance for impact study on similarity threshold θ, for all websites

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

True/Predicted	HTML	Target	Neither
HTML	732020	155	0
Target	446	255712	0
Neither	25423	11280	0

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

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

Table 12: 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

E SUPPLEMENTARY MATERIAL FOR SECTION 6 (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 Section 2). Moreover, the websites that such crawlers are targeting, especially forums and blogs, are not the most natural candidates for extracting large amounts of targets; let alone focusing on the acquisition of *trustworthy* targets.

(3) *Hidden-* or *deep*-Web crawlers. They postulate that the Web might not be accessible just following hyperlinks on HTML pages, but also behind interfaces where an interaction with the user is required. Most of the time, these interactions are forms to be filled and sent, query interface, search interface, etc. [HRR19] presenting most state-of-the-art deep-Web crawlers, as well as a framework allowing comparison of such crawlers regarding a wide range of aspects. Our approach does not cover this type of crawling, and therefore constitutes a natural extension to our work (as discussed in Section 7). 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 Section 5.1), these targets are generally not accessible through a portal, but rather by navigating through the links.

(4) *Incremental* or *revisit policy-based* crawlers. They postulate that the Web is not a static collection of pages, but a dynamic one: therefore the pages of a given website are likely to evolve over time. Thus, there is a need for revisiting some of the pages. The challenge here is to be able to minimize the number of revisited pages while maximizing the retrieval of updated content. [CGM00]

presents the most important challenges regarding incremental crawling and how they should be influenced by the evolution of the Web. [Sig05] presents an incremental version of the *Heritrix* project [MSR+04], *Internet Archive*'s[9] 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.

REFERENCES FOR THE APPENDIX

[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 Luís Sarmento. 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.

[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.

[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.

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

[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.

[9] https://archive.org/