# Adaptive Crawling Driven by Structure-Based Link Classification

Muhammad Faheem Pierre Senellart
Institut Mines–Télécom
Télécom ParisTech; CNRS LTCI Paris, France
firstname.lastname@telecom-paristech.fr

July 28, 2014

User-generated content on the Web has been growing at a fast pace, and has become an important data resource for which much effort has been put into developing wrappers (i.e., programs that extract content from HTML pages). However, current wrapper generation systems focus on individual Web pages, while retrieving Web pages to extract from often remains a manual or supervised process. Generic crawling approaches are inefficient in that they cannot distinguish among various page types, and cannot target content-rich areas of a Web site. In this paper, we study the problem of efficient unsupervised Web crawling of content-rich Web pages. We propose ACEBot (Adaptive Crawler Bot for data Extraction), a structure-driven crawler that utilizes the inner structure of the pages and guides the crawling process based on the importance of their content. ACEBot works in two phases: in the *offline* phase, it constructs a dynamic site map (limiting the number of URLs retrieved), learns a traversal strategy based on the importance of *navigation patterns* (selecting those leading to valuable content); in the *online* phase, ACEBot performs massive downloading following the chosen navigation patterns. Extensive experiments over a large dataset illustrate the effectiveness of our system. ACEBot ignores duplicate and invalid pages and crawls only important content with high precision and recall. Our system makes 5 times fewer HTTP requests as compared to a generic crawler, without compromising on effectiveness.

## 1 Introduction

**The Web and User-Created Content** The incredible growth of the World Wide Web has revolutionized the information age. Now the Web has become the largest data repository available to humankind, and several efforts have been made to utilize this resource. The Web is widespread (but ephemeral) and many social activities are happening there everyday. User-created content (UCC) has been growing at a very fast pace [19]. The cultural effects of this social phenomenon are also significant. Indeed, the Web has become a vast digital cultural artifact that needs to be preserved. But, unfortunately, important parts of our cultural heritage have already disappeared; as an example, the first ever Web page (its first version) that Tim Berners-Lee wrote back in 1990 has reportedly been lost [4]. Certainly, Web preservation is a cultural and historical necessity for preserving valuable information for historian, journalists, or social scientists. Indeed, this is an objective of *Web archiving* [17], which deals with selecting, crawling, preserving, and ensuring long-term access to historical Web content.

A large part of Web content (especially user-created content) belong to Web sites powered by content management systems (CMSs) such as vBulletin, phpBB, or WordPress [11]. The presentation layer of these CMSs use predefined templates (which may include left or right sidebar of the Web document, header and footer, navigation bar, main content, etc.) for populating the content of the requested Web document from an underlying database.

These templates control the layout and appearance of the Web pages. A study [11] has found that 40-50% content on the Web (in 2005) was template-based, growing at the rate of 6-8% per year. Depending on the request, the CMSs may use different templates for presenting information; e.g., for blogs, the *list of posts* type of page may use a different template than the *single post* Web page that also include comments. These template-based Web pages form a meaningful structure that mirror the implicit logical relationship between Web content across different pages within a Web site. Many templates are used by CMSs for generating different type of Web pages. Each template generates a set of Web pages (e.g., list of blog posts) that share the common structure, but differ in terms of content. These templates are consistently used across different regions of Web site. More importantly, in a given template (say, list of posts), links leading to a specific kind of content (say, individual posts) usually share a common layout and presentation properties.

**Crawling the Web** Many public and private institutions are archiving large collection of public content such as *Internet Archive*[1], *Internet Memory*[2], or a variety of national libraries around the world. Robots of commercial search engine such as `Google` and `Bing` crawl information from UCC on the Web to improve the quality of search results. Though Web content exists in large amount, due to limited bandwidth, storage, or indexing capabilities, only a small fraction of the content can be harvested by Web crawlers. This is true for crawlers of institutions with limited resources (e.g., the national library of a small country). This is even true for a company such as `Google`, that has discovered more than a trillion unique URLs [1] in the frontier, but indexed around 40 billions Web pages (as of June, 2013.) [9]. This suggests a need to develop a crawling strategy that not only effectively crawls Web content from template-based Web sites, but also efficiently minimizes the number of HTTP requests by avoiding non-interesting Web pages.

A generic Web crawler performs inefficient crawling of the Web sites. It crawls the Web with no guarantee of content quality. A naive *breadth-first* (shallow) strategy cannot ensure access to all content-rich pages, while a simple *depth-first* (deep) crawl may fetch too many redundant and

[1] `http://archive.org/`
[2] `http://internetmemory.org/`

invalid pages (e.g., may login failure pages), in addition to the possibility of falling into *robot traps*. Thus, a generic methodology crawls a set of Web pages that may lack real interest for a given user or application. An ideal crawling approach should solve the two following problems: first what kind of Web pages are *important* to crawl (to avoid redundant and invalid pages); and second which *important* links should be followed and what navigation patterns are required on the Web site?

**Towards an Intelligent Crawler** We introduce in this article an intelligent crawling technique that meet the above-stated criteria. We propose a *structure-driven* approach that is more precise, effective, and achieve higher quality level, without loss of information. It guides the crawler towards content-rich areas: this is achieved by learning the best traversal strategy (a collection of important navigation patterns) during an offline phase that ultimately guides the crawler to crawl only content-rich Web pages during online phase.

Our structure-driven crawler, ACEBot, first establishes connections among Web pages based on their root-to-link paths, then rank paths according their importance (i.e., root-to-links paths that lead to content-rich Web pages), and further learns a traversal strategy for bulk-downloading of the Web site. Most existing efforts on Web crawling only utilize hyperlink-related information such as URL pattern and anchor text [6, 18] to design a crawling strategy. Such approaches ignore the relationships among various Web pages and may judge the same kind of hyperlink appearing on different pages independently. Our main claim is that structure-based crawling strategy not only cluster Web pages which require similar crawling actions, but also helps to identify duplicates, redundancy, boilerplate, and plays as well an important role in prioritizing the frontier. Web pages that requires similar navigation patterns are believed to have similar types of content and to share the same structure.

We first present our model in Section 2. The algorithm that ACEbot follows is then presented in detail in Section 3. We finally present experiments in Section 4. Before concluding, we discuss the related work in Section 5.

## 2 Model

In this section, we formalize the model of our proposed approach: we see the Web site to crawl as an abstract directed graph, that is rooted (typically at the homepage of a site), and where edges are labeled (by structural properties of the corresponding hyperlink). We first consider the abstract problem, before explaining how we turn a Web site to crawl into an instance of this problem.

### 2.1 Formal Definitions

We fix countable sets of *labels* $\mathscr{L}$ and *items* $\mathscr{I}$. Our main object of study is the graph to crawl:

**Definition 1.** *A* rooted graph *is a 5-tuple* $G = (V, E, r, \iota, l)$ *where* $V$ *is a finite set of vertices,* $E \subseteq V^2$ *is a set of directed edges (potentially including loops),* $r \in V$ *is the* root *of the graph,* $\iota : V \to 2^{\mathscr{I}}$ *assigns a set of items to every vertex, and* $l : E \to \mathscr{L}$ *assigns a* label *to every edge.*

Here, items serve to abstractly model the interesting content of Web pages. We naturally extend the function $\iota$ to a set of nodes $X$ from $G$ by posing: $\iota(X) = \bigcup_{u \in X} \iota(u)$.

We introduce the standard notion of paths within the graph:

**Definition 2.** *Given a rooted graph $G = (V, E, r, \iota, l)$ and vertices $u, v \in V$, a* path *from $u$ to $v$ is a finite sequence of edges $e_1 \dots e_n$ from $E$ such that there exists a set of nodes $u_1 \dots u_{n-1}$ in $V$ with:*

- *$e_1 = (u, u_1)$;*
- *$\forall 1 < k < n, e_k = (u_{k-1}, u_k)$;*
- *$e_n = (u_{n-1}, v)$.*

*The* label *of the path $e_1 \dots e_n$ is the word over $\mathscr{L}$ $l(e_1) \dots l(e_n)$.*

Critical to our approach is the notion of *navigation pattern* that uses edge labels to describe which paths to follow in a graph:

**Definition 3.** *A* navigation pattern *$p$ is a regular expression over $\mathscr{L}$. Given a graph $G = (V, E, r, \iota, l)$, the result of applying $p$ onto $G$, denoted $p(G)$, is the set of nodes $u$ such that there exists a path from $r$ to $u$ with a label* a prefix of a word *in the language defined by $p$.*
*We extend this notion to a finite set of navigation patterns $P$ by letting $P(G) := \bigcup_{p \in P} p(G)$.*

Note that we require only a *prefix* of a word to match: a navigation pattern does not only return the set of pages whose path from the root matches the regular expression, but also pages on those paths. For instance, consider a *path* $e_1 \dots e_n$ from $r$ to a node $u$, such that the navigation pattern $p$ is the regular expression $l(e_1) \dots l(e_n)$. Then, the result of executing navigation pattern $p$ contains $u$, but also all pages on the path; more generally, $p$ returns all pages whose path from the root matches a prefix of the expression $l(e_1) \dots l(e_n)$.

Navigation patterns are assigned a score:

**Definition 4.** *Let $G = (V, E, r, \iota, l)$ be a rooted graph. The* score *of a finite set of navigation patterns $P$ over $G$, denoted $\omega(P, G)$ is the average number of distinct items per node in $P(G)$, i.e.:*

$$\omega(P, G) = \frac{|\iota(P(G))|}{|P(G)|}.$$

We can now formalize our problem of interest: given a rooted graph $G$ and a collection of navigation patterns $\mathscr{P}$ (that may be all regular expressions over $\mathscr{L}$ or a subclass of regular expressions over $\mathscr{L}$), determine the set of navigation patterns $P \subseteq \mathscr{P}$ of maximal score over $G$: $\arg\max_{P \subseteq \mathscr{P}} \omega(P, G)$.. Unfortunately, we will show this problem is NP-hard. First, determining if the score is at least a given number is intractable:

**Proposition 1.** *Given a graph $G$, a collection of navigation patterns $\mathscr{P}$, and a constant $K$, determining if there exists a finite subset $P \subseteq \mathscr{P}$ with score over $G$ at least $K$ is an NP-complete problem.*

*Proof.* First, we argue that our problem is in NP, since given a graph $G$, a collection of navigation patterns $\mathscr{P}$ and a constant $K$, we can non-deterministically guess a subset $P \in \mathscr{P}$, compute its score over $G$, and check whether this score is at least $K$. The running time is polynomial.

For hardness, we apply a reduction from Set Cover. Given an instance of Set Cover (i.e., given a finite set $U$, a collection of sets $(S_i)_{1 \leq i \leq m}$ of elements of $U$, and an integer $k$, determine whether there exists a set $C \subseteq \{1,2\ldots m\}$ such that $|C| \leq k$ and $\bigcup\limits_{i\in C} S_i = U$), we construct the instance of our problem.

The set of items $\mathscr{I}$ is a superset of $U$; the set of labels has $m+1$ distinct labels $l_0, l_1 \ldots l_m$. We first construct $G = (V,E,r,\iota,l)$ as follows: $V = \{x_0,x_1,\ldots,x_N,s_1,\ldots,s_m\}$ is a set of $N+1+m$ nodes (where $N$ is an integer that we will define later on). $E$ consists of the following edges:

- For all $1 \leq i \leq N-1$, an edge $(x_i,x_{i+1})$ with label $l(x_i,x_{i+1}) = l_0$.
- For all $1 \leq i \leq m$, an edge $(x_N,s_i)$ with label $l(x_N,s_i) = l_i$.

The root $r$ is $x_0$. The mapping $\iota$ is defined by $\iota(x_i) := \emptyset$ for $0 \leq i \leq N$ and $\iota(s_i) := S_i$ for $1 \leq i \leq m$. For $\mathscr{P}$ we take the set $\{\, l_0^* l_i \mid 1 \leq i \leq m \,\}$. The score of a pattern $l_0^* l_i \in \mathscr{P}$ is $\omega(l_0^* l_i, G) = \frac{|S_i|}{N+1}$. For $P_C = \{\, l_0^* l_i \mid i \in C \,\}$ with $C \subseteq \{1,2\ldots m\}$, $\omega(P_C,G) = \frac{\left|\bigcup\limits_{i\in C} S_i\right|}{|C|+N}$. See Figure 1 for an illustration of the construction.

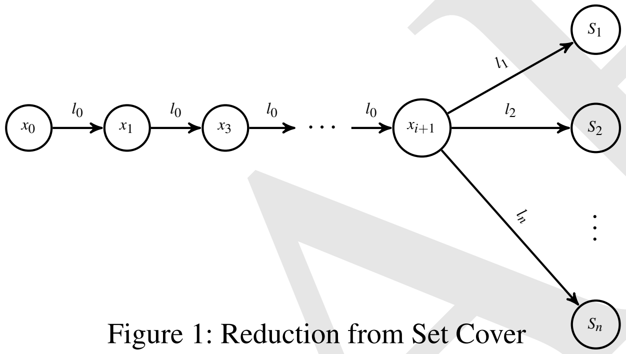


Figure 1: Reduction from Set Cover

We now show that there exists a subset $P \subseteq \mathscr{P}$ with score over $G$ at least $\frac{|U|}{k+N}$ if and only if the set cover instance has a solution of size at most $k$.

$\Leftarrow$ Assume the set cover instance has a solution of size at most $k$. Then there is a set $C \subseteq \{1,2\ldots m\}$ such that $|C| \leq k$ and $\bigcup_{i\in C} S_i = U$. The score of the set of patterns $P_C$ is: $\omega(P_C,G) = \frac{\left|\bigcup\limits_{i\in C} S_i\right|}{|C|+N} \geq \frac{|U|}{k+N}$.

$\Rightarrow$ Let $P_C$ such that $\omega(P_C,G) \geq \frac{|U|}{k+N}$. We distinguish two cases. First, assume $\bigcup\limits_{i\in C} S_i = U$. Then $\frac{\left|\bigcup\limits_{i\in C} S_i\right|}{|C|+N} \geq \frac{|U|}{k+N}$ implies that $|C| \leq k$ and $(S_i)_{i\in C}$ is a solution to the set cover problem. Otherwise, we have $\left|\bigcup\limits_{i\in C} S_i\right| \leq |U|-1$. We will show that this leads to a contradiction for a well-chosen value of $N$ (observe that the choice of $N$ has been left fully open until now). We take $N = m \times (|U|-1)+1$ (as $U$ is an input to the set cover problem, having $N+1+m$ nodes in $G$ still results in a polynomial-time construction). We have:

$$\omega(P_C, G) = \frac{\left|\bigcup_{i\in C} S_i\right|}{|C| + N} \leq \frac{|U| - 1}{|C| + m \times (|U| - 1) + 1}$$
$$\leq \frac{|U| - 1}{m \times (|U| - 1) + 1} = \frac{1}{m + \frac{1}{|U|-1}}$$

whereas:

$$\omega(P_C, G) \geq \frac{|U|}{k + N} = \frac{|U|}{k + m \times (|U| - 1) + 1}$$
$$\geq \frac{|U|}{m + m \times (|U| - 1) + 1}$$
$$= \frac{|U|}{m|U| + 1} = \frac{1}{m + \frac{1}{|U|}}$$
$$> \frac{1}{m + \frac{1}{|U|-1}}.$$

We reach a contradiction that shows that, necessarily, $\bigcup_{i\in C} S_i = U$.

This concludes the reduction. The construction we used is polynomial time: the graph $G$ is at most quadratic in the size of the original set cover instance (the quadratic explosion comes from the choice of $N$). $\square$

A simple corollary of this proposition shows the hardness of determining if a set of navigation patterns has optimal score:

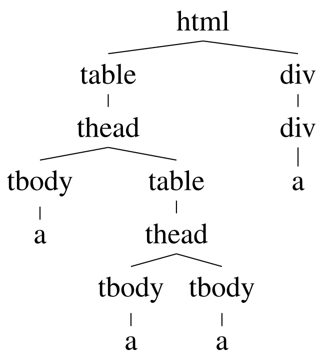
**Corollary 1.** *Given a graph $G$ and a collection of navigation patterns $\mathscr{P}$, determining if one finite subset $P \subseteq \mathscr{P}$ has maximal score over $G$ is a coNP-complete problem.*

*Proof.* First, We argue that determining whether one subset $P \subseteq \mathscr{P}$ has maximal score over $G$ is in coNP: to determine whether $P$ is *not* maximal, guess anther subset of navigation patterns $P'$, compute its score in polynomial time, and check whether its less than the score of $P$.

Let's turn to the hardness. We reduce from the previous proposition. Let $G = (V, E, r, \iota, l)$ be a graph, $\mathscr{P}$ be a collection of navigation patterns, $K$ a constant. Let $n = |V|$. Without loss of generality, we assume that $\iota(r) = \emptyset$ (otherwise, just add another dummy root, and update other parameters accordingly). Let $q = \arg\min_{1\leq i\leq n, \lfloor Ki\rfloor \neq Ki} K - \frac{\lfloor Ki\rfloor}{i}$.

We construct a graph $G' = (V', E', r, \iota', l')$ as an extension of $G$: $V' = V \cup \{y_0, \dots, y_q\}$, $E' = E \cup \{(r, y_0), (y_0, y_1), \dots, (y_{q-1}, y_q)\}$, $r$ is the same, $\iota'$ is the same on nodes of $V$, $\iota(y_i) = \emptyset$ for $0 \leq i < n$, and $\iota(y_n)$ is a set of $\lfloor K \times q\rfloor$ fresh items. The label of the fresh edges is set to a fresh label $\lambda$. For the collection of navigation patterns, we take $\mathscr{P}' = \mathscr{P} \cup \{\lambda^*\}$.

Now, we claim that there exists a subset $P \subseteq \mathscr{P}$ with score over $G$ at least $K$ if and only if $\{\lambda^*\}$ has maximal score over $G'$. Note that $\omega(\{\lambda^*\}, G') = \frac{\lfloor Kq\rfloor}{q} < K$. First assume that such a subset with score over $G$ at least $K$ exists. But $\omega(G', P) = \omega(G, P) \geq K$. Then $\{\lambda^*\}$, with score



(a) DOM tree representation for selected Web links

| | |
|---|---|
| $l_1$ | html-table-thead-table-thead-tbody-a |
| $l_2$ | html-div-div-a |
| $l_3$ | html-table-thead-tbody-a |

(b) root-to-link paths

Figure 2: A sampled Web page represented by a subset of the root-to-link paths in a corresponding DOM tree representation.

$< K$, cannot have maximal score. Conversely, assume such a subset does not exist. Let $P$ be any subset of $\mathscr{P}$. By definition of $q$, $\omega(G',P) = \omega(G,P) \leq \frac{\lfloor Kq \rfloor}{q} = \omega(G',\{\lambda^*\})$ and thus $\{\lambda^*\}$ has maximal score. $\square$

Thus, there is no hope in obtaining of efficiently obtaining an optimal set of navigation patterns. In Section 3, we will develop a greedy approach to the selection of navigation patterns.

## 2.2 Model generation

We now explain how we consider crawling of a Web site in the previously introduced abstract model.

A Web site is any HTTP-based application, formed with a set of interlinked Web pages that can be traversed from some base URL, such as `http://www.wsdm-conference.org/`. The base URL of a Web site is called the entry point of the site. For our purpose, we model a given Web site as a directed graph (see Definition 1), where the base URL becomes the root of the graph. Each vertex of the graph represents a distinct Web page and, following Definition. 1, a set of items is assigned to every vertex. In our model, the items are all *distinct 2-grams* seen for a Web page. A 2-gram for a given Web page is a contiguous sequence of 2 words within its HTML representation. The set of 2-grams has been used as a summary of the content of a Web page [10]; the richer a content area is, the more distinct 2-grams. The set of items associated to each vertex plays an important role in the scoring function (see Definition 4), which eventually leads to selecting a set of Web pages for crawling.

A web page is a well-formed HTML document and its Document Object Model (DOM [24]) specifies how objects (i.e., texts, links, images, etc.) in a Web page are accessed. Hence, a *root-to-link* path is a location of the link (i.e., <a> HTML tag) in the corresponding DOM tree [24]. Figure 2a shows a DOM tree representation and Figure 2b illustrates its *root-to-link* path for a sample Web page.

Following the Definition 1, each edge of the graph is labeled with a *root-to-link* path. Assume there is an edge $e(u,v)$ from vertex $u$ to $v$, then a label $l(e)$ for edge $e$ is the *root-to-link* path of the hyperlink pointing to $v$ in vertex (i.e., Web page) $u$. Navigation patterns will thus be (see Definition 3) regular expression over root-to-link paths.

Two Web pages reachable from the root of a Web site with paths $p_1$ and $p_2$ whose label is the same are said to be *similar*.

Consider the scoring of a navigation pattern (see Definition 4). We can note the following:

- the higher the number of requests needed to download pages comprised by a navigation pattern, the lower the score;
- the higher the number of distinct n-grams in pages comprised by a navigation pattern, the higher the score.

## 3 Deriving the Crawling Strategy

In this section, we detail the crawling strategy for any given Web site. We begin by illustrating our approach with a simple example and then formally describe our unsupervised crawling technique.

### 3.1 Simple Example

Consider the homepage of a typical Web forum, say `http://forums.digitalspy.co.uk/`, as the entry point of the Web site to crawl. This Web page may be seen as a two different regions. There is a region with headers, menus and templates, that are presented across several Web pages, and is considered as a non-interesting region from archiving perspective. The other region at the center of the Web page is a content-rich area and required to be be archived. Since these pages are generated by a CMS (vBulletin in this particular case), the underlying templates have a coherent structure across similar Web pages. Therefore links contained in those pages obey regular formating rules. In our example Web site, the links leading to blog posts and the messages within an individual post should have some layout and presentational similarities.

Figure 2a presents a subset of the DOM tree for the example entry point Web page and its *root-to-link* paths are shown in Figure 2b. Figure 4a shows a truncated version of the generated graph (see Section 2.2) for the corresponding site. Each vertex represents a unique Web page in the graph. These vertices are connected through directed edges, labeled with *root-to-link* paths. Each vertex of the graph is assigned a number of distinct 2-gram seen for the linked Web page (e.g., 3,227 distinct 2-grams seen for $p_3$). Furthermore, the set of Web pages (i.e., vertices) that share the same path (i.e., edge label) are clustered together (see Figure 4b). The newly clustered vertices are assigned a collective 2-gram set seen for all clustered Web pages. For instance, the clustered vertex $\{p_3, p_4\}$ has now 5,107 distinct 2-gram items. After clustering, all possible navigation patterns are generated for the graph. This process is performed by traversing the directed graph. Table 1 exhibits all possible navigation patterns. Afterwards, each navigation pattern (possibly combination of *root-to-link* path) is assigned a score. The system does not compute the score for any navigation pattern that does not lead the crawler from the entry point of the Web site. Therefore, the system has ignored the navigation pattern $l_4$ (shown underlined). Here the score $\underline{2600}$ for navigation pattern $l_4$ is computed just for the sake of understanding, in

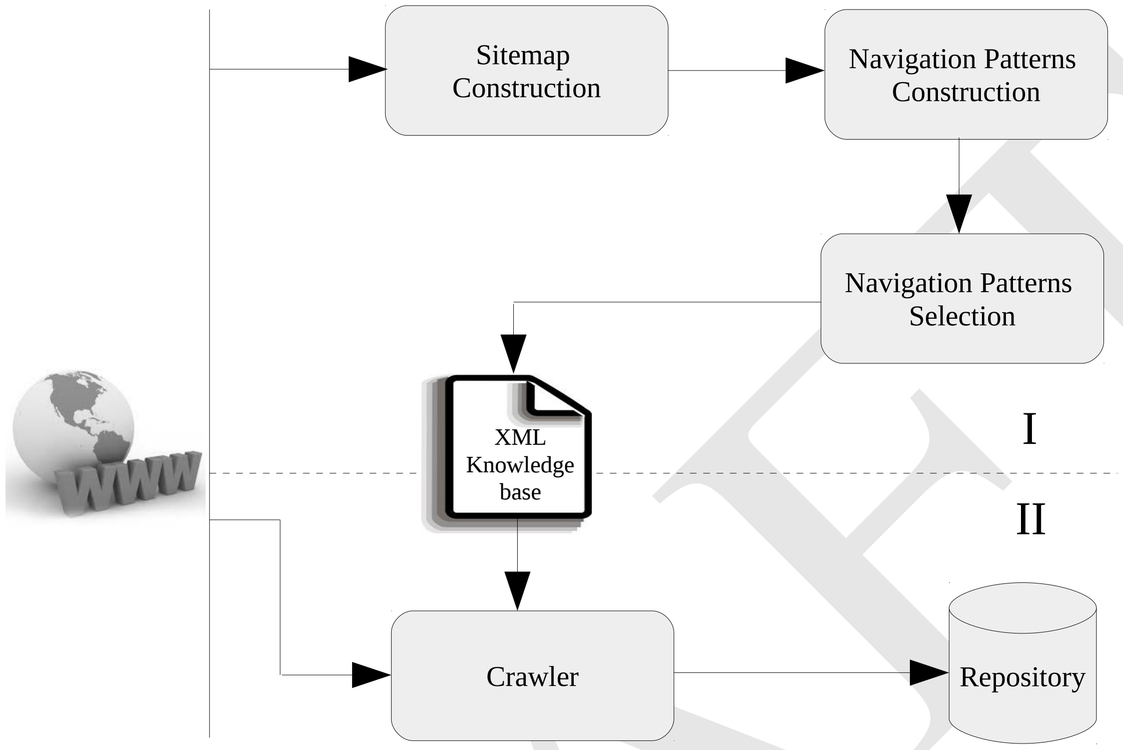

Sitemap Construction
Navigation Patterns Construction
Navigation Patterns Selection
XML Knowledge base
I
II
WWW
Crawler
Repository

Figure 3: Architecture of ACEBot, which consists of two phases: (I) offline sitemap construction and navigation patterns selection; and (II) online crawling.

practice, the system will not compute it. Once all possible navigation patterns are scored then the navigation pattern with highest score is selected (since highest score ensures the archiving of core-contents). Here, the navigation pattern $l_1$ is selected (not $l_4$).

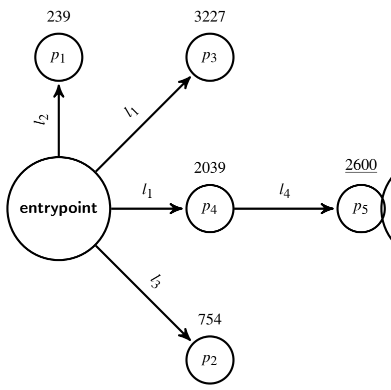
The process of assigning the score to the navigation patterns keeps going after each selection for navigation patterns not selected so far. Importantly, 2-gram items for already selected vertices are not considered again for non-selected navigation patterns. Therefore, in the next iteration, the navigation pattern $l_1 l_4$ does not consider items from Web pages of the $l_1$ navigation pattern. The process of scoring and selecting ends when no interesting navigation pattern is left to follow.

Since we believe Web sites that belong to the same CMSs may enjoy the common templates, learned set of navigation patterns may work for several such Web sites.

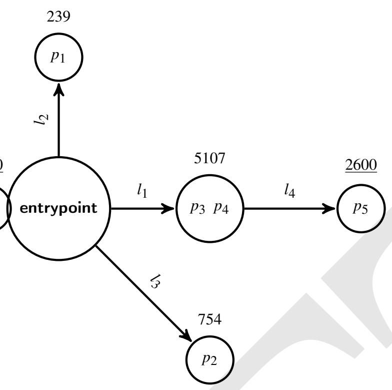
## 3.2 Detailed Description

In this section, we detail the process of the unsupervised structure-driven crawler (ACEBot) proposed in our approach. First, we discuss the sitemap construction module. Then we present the navigation pattern construction and selection modules.

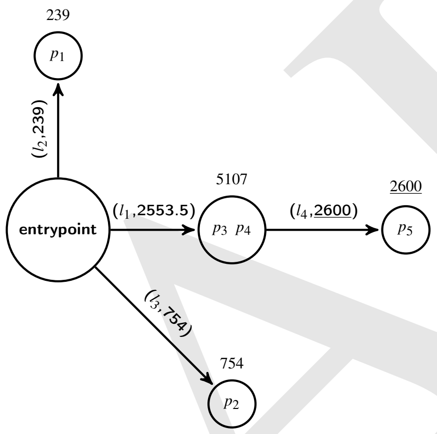
ACEBot (see Figure 3) mainly consists of two phases: offline and online phase. The aim of the offline phase is to first construct the sitemap and cluster the vertices that share the similar edge label. Then a set of crawling actions (i.e., best navigation patterns) are learned to guide the



(a) Before clustering



(b) After clustering



(c) After scoring

Figure 4: Truncated Site Model with scores for the exemplary Web site (see Figure 2b for full labels)

massive crawling in online phase.

Algorithm 1 gives a high-level view of the navigation pattern selection mechanism for a given entry point (i.e., home page). Algorithm 1 has six parameters. The entry point *r* is the home page of a given Web site. The Boolean value of the parameter *d* specifies whether the sitemap of the Web site should be constructed dynamically. The argument *k* defines the depth (i.e., level or step) of navigation patterns to explore. The Boolean *ac* specifies whether to limit the continuity of navigation patterns to a fixed value of 3. For instance, for page-flipping navigation pattern `/(/html/body/div[contains(@id,"navigation")]/a/@href{click/})+`, the + indicate that the action will be executed atleast once (i.e., $\{1,*\}$, where 1 is a lower limit). When Boolean value true is passed to *ac*, then the $\{1,3\}$ bound will be used. We have restricted the upper bound to 3; this will ensure that the navigation pattern that form the continuity of similar pages (e.g., page flipping navigation pattern) is selected. If we do not consider the upper bound 3 but just 1, then there are chances that page-flipping navigation patterns may be neglected because of lower score. Such navigation patterns may consist of a single page, and score based on one page may not lead to the right selection. This may also select other non important navigation patterns (that are not page-flipping actions). The experiments have shown that there were cases when weak navigation patterns were selected, but were avoided when limiting the continuity constraint to

| NP | total 2-grams | distinct 2-grams | score |
|---|---|---|---|
| $l_4$ | 2600 | 2600 | <u>2600</u> |
| $l_1$ | 5266 | 5107 | 2553.5 |
| $l_1, l_4$ | 7866 | 7214 | 2404.7 |
| $l_3$ | 754 | 754 | 754 |
| $l_2$ | 239 | 239 | 239 |

Table 1: Navigation patterns with score for the example of Figure 4

```
Input: entry point $r$, dynamic sitemap $d$, navigation pattern continuity limit $ac$,
       navigation-step $k$, a set of attributes $a$, completion ratio $cr$
Output: a set of selected navigation patterns $SNP$
$siteMap \leftarrow generateSiteMap(r, d)$;
$clusteredGraph \leftarrow performClustering(siteMap)$;
for $r \in R$ do
    $navigationPatterns \leftarrow getNavigationPatterns(r, clusteredGraph, k, ac, a)$;
    $NP \leftarrow updateNavigationPatterns(navigationPatterns)$;
while not $cr$ do
    $topNP \leftarrow getTopNavigationPattern(NP, SNP)$;
    $SNP \leftarrow addToSelectedNP(topNP)$;
    $NP \leftarrow removeSubNavigationPatterns(topNP)$;
```

**Algorithm 1:** Selection of the navigation patterns

3. The argument *a* passes the set of attributes (e.g., id, and class) that should be considered when constructing navigation patterns. *cr* sets the completion ratio, and the process of selecting navigation patterns ends when this criteria met.

The goal of the offline phase is to obtain useful knowledge for a given Web site based on a few sample pages. The sitemap construction is the foundation of the whole crawling process. The quality of sampled pages is important to decide whether learned navigation patterns target the content-rich part of a Web site. We have implemented a double-ended queue (similar to the one used in [5]), and then fetched the Web pages randomly from the front or end. We have limited the number of sampled pages to 3000, and detailed experiments (See Section 4) have found that the sample restriction was enough to construct the sitemap of any considered Web site. The *generateSiteMap* procedure takes a given entry point as parameter and returns a sitemap (i.e., site model) as described in Definition 1. The graph vertices are Web pages and edges between these vertices are *root-to-link* (location of the Web link) paths. For each Web page, the procedure analyze the links (i.e., <a> HTML tags) and their location within the DOM tree. It also computes distinct 2-grams (contiguous sequence of 2 consecutive words) seen for each Web page. A unique vertex is formed for each Web page and it is connected with its links via directed outgoing edges (See Figure 4a). Further, each Vertex of the graph is labeled with a number of distinct 2-grams seen for its Web page, and each arc leading to link is labeled with *root-to-link* path(see Figure 2b, 4a).

Intuitively, we assume that few Web links (i.e., <a> HTML tags) share the *root-to-link* paths within a Web page. Therefore, a vertex may have several destination vertices who share the same edge label. The importance of a specific navigation pattern (i.e., *root-to-link* path) ensures that the destination nodes hold the content-rich area of a Web site. This eventually helps to estimate the importance of the crawled Web pages. We cluster Web pages which share the similar navigation patters, and thus it is first approximation to approach the problem of discovering similar Web pages.

The procedure *performClustering* in algorithm 1 clusters the vertices with similar edge labels. It performs breadth-first traversal over the graph, starting from each root till the last destination vertex. For instance, Figure 4b, vertex $p_3$ and $p_4$ share the label $l_1$ and thus are clustered together. The 2-gram measure is also computed for each clustered vertex. More precisely, the clustering of the similar nodes is performed in two way:

- Clustering the destination vertices that share the edge label. For instance, list of blog posts where label $l_2$ is shared among several vertices.
- Let vertex $v'$ has an incoming edge from vertex $v$ with label $l_1$, and also vertex $v'$ has an outgoing edge to vertex $v''$ with similar label $l_1$. Since $v'$ and $v''$ share edge label, therefore these vertices will be clustered. For instance, page-flipping links (For instance, post messages that may exist across several pages) usually has the same *root-to-link* path. These type of navigation patterns end with + (e.g., `/html/body/div[contains(@class," navigation")])+`), that indicate that crawling action should be performed more than once on similar Web pages during online phase. These type of actions has two advantages: learn a navigation pattern that hold the portion of Web site with many pages, and give enough evidence of it selection; inform the online phase to execute the action several times.

Moreover, the clustering phase also ensures that redundant Web pages do not play a role in selection of the best navigation patterns. Since we are computing the number of distinct 2-gram items for each Web page, therefore the clustering phase does not group any new Web page which has similar 2-gram items to the existing (already clustered) Web page. For instance, the login page may always has the similar 2-grams items.

Once the graph is clustered, the *getNPWithScore* extracts all possible navigation patterns for each root vertex $r \in R$. The procedure takes three parameters *clusteredGraph*, $r$, and $k$ as input. The $k$ parameter limits the number of navigation steps for a specific root vertex $r$. The procedure generates the navigation patterns using depth-first traversal approach where depth is limited to $k$ (i.e., number of navigation-steps). Since the aim of our approach is to find a set of navigation patterns that lead the crawler from entry point of the Web site to the interesting pages, therefore, here we only consider the navigation patterns that has root vertex as a start node. Hence, a set of navigation patterns are generated, starting from root vertex till the $k$ number of navigation-steps. This step will be performed for each root vertex and *updateNavigationPatterns* will update the set of navigation patterns *NP* accordingly.

The *getTopNavigationPattern* procedure returns a top navigation pattern on each iteration. The procedure takes two parameters *NP* (a set of navigation patterns), and *SNP* (a set of selected navigation patterns) as input. This procedure applies the subset scoring function (see Definition 4) and computes the score for each navigation pattern according to the rewritten scoring function as defined in section 2.2. The $items(NP)$ is computed by counting the total number of distinct 2-grams words seen for all vertices that share the navigation pattern *NP*. The size of the navigation pattern *NP* (i.e., $size(NP)$) is the total number of vertices that shares the *NP*. The *SNP* parameter is passed to the procedure to ensure that only new data rich areas are identified. Therefore, the scoring function does not consider the Web pages (i.e., clustered vertices), which are already discovered and hence does not take into account the score associated with them to evaluate new vertices. More precisely, assume the $l_1l_2$ navigation pattern is already selected. Now the scoring function for navigation pattern $l_1l_2l_3$ does not take into account the score for navigation pattern $l_1l_2$, but only $l_3$ score will play a role in its selection. Eventually, it guarantee that the system always select the navigation patterns with newly discovered Web pages with valuable content. The *removeSubNavigationPatterns* procedure removes all the sub navigation patterns. For instance, if navigation pattern $l_1l_4l_5$ is selected first then there is no need to evaluate the score for the navigation pattern $l_1l_4$ (if not already selected), since it will not guide to the new Web pages.

The redundant cluster (i.e., any two navigation patterns that has similar 2-gram items) will also not be selected by the *getTopNavigationPatterns* procedure. For instance, for blog-like Web sites, a same blog post may accessible by tag, year, etc. Consider navigation patterns $l_1l_5$ and $l_1l_6$ lead the crawler to redundant Web pages, and if the $l_1l_5$ is already selected then the $l_1l_6$ will not be selected. The $l_1l_6$ will get low score since it does not learn the new 2-grams items. Similarly, any navigation pattern that has redundant pages inside will have *few* distinct 2-grams items, and therefore will get a low score from the scoring function and thus will have less probability of its selection.

The selection of navigation patterns ends when all navigation pattern from the set *NP* are selected or when content-rich Web pages met with the criteria (i.e. *completion ratio cr* condition satisfied). In our implementation, we have set the criteria to the 95% coverage of distinct 2-gram items seen for given entry point. The example of such a navigation pattern is:

```
doc("www.rockamring-blog.de/index.html")
/(/html/body/div[contains(@id,"wrapper")]/div[contains(@id,"navigation")]/a//@href{click
/})+
/(/html/body/div[contains(@id,"wrapper")]/div[contains(@id,"post-")]/a/@href{click/})
```

We use here a restriction of the OXPath [21] language (see Figure 5) for the syntax navigation patterns.

When selected navigation patterns reaches the *cr* coverage of the total number of distinct 2-grams seen on the sitemap, the system will call the online phase and feed the selected navigation patterns to the crawler for massive online crawling.

## 4 Experiments

In this section, we present the experimental results of our proposed system. We compare the performance of ACEBot with AAH [10] (our previous work, that relies on a hand-written description of given Web sites), iRobot [5] (a system of the literature dedicated to the efficient crawling of Web forums), and GNU wget[3] (a traditional Web crawler), in terms of efficiency and effectiveness.

The open-source ACEBot code and the experimental dataset (a list of sites) are freely available at `http://perso.telecom-paristech.fr/~faheem/acebot.html`.

[3]http://www/gnu.org/software/wget/

```
⟨expr⟩      ::= "doc" "(" ⟨url⟩ ")" (⟨estep⟩)+

⟨estep⟩     ::= ⟨step⟩ | ⟨kleene⟩
⟨step⟩      ::= "/" "("⟨action⟩ ")"
⟨kleene⟩    ::= "/" "(" ⟨action⟩ ")" ( "*" | ⟨number⟩ )
⟨action⟩    ::= ("/" ⟨nodetest⟩)+ "{click/}"
⟨nodetest⟩  ::= tag | "@" tag
```

Figure 5: BNF syntax of the OXPath [21] fragment used for navigation patterns. The following tokens are used: *tag* is a valid XML identifier; <url> follows the w3c BNF grammar for URLs is located at `http://www.w3.org/Addressing/URL/5_BNF.html#z18`.

## 4.1 Experiment Setup

To evaluate the performance of ACEBot at a Web-scale, we have carried out the evaluation of our system in various settings. Here, first we describe the dataset and performance metrics and different settings of our proposed algorithm.

**Dataset** We have selected 50 Web sites (totaling nearly 2 million Web pages) with diverse characteristics, to analyze the behavior of our system for small Web sites as well as for Web-scale extraction. The evaluation of ACEBot for different content domains is performed to examine its behavior in various situations. We consider Web sites of type blogs, news, music, travel, and books. We crawled 50 Web sites with both wget (for a full, exhaustive crawl), and our proposed system. To compare the performance of ACEBot with AAH, 10 Web sites (nearly 0.5 million Web pages) were crawled with both ACEBot and AAH.

**Site map** In offline phase, the site map of a given Web site is constructed either from the whole mirrored Web site or from a smaller collection of randomly selected sample pages. The mechanism for random (i.e., dynamic) selection of sample pages for a given entry point is described in Section 3.2. We found that ACEBot requires a sample of 3,000 pages to achieve optimal crawling quality, comparable to what was done for iRobot [5] (1,000 pages) and a supervised structure driven crawler [23] (2,000 pages). We present in Figure 6 the variation of performance of ACEBot (in term of number of crawled distinct n-grams) as the number of sampled pages per site increases, shown averaged over 10 Web sites having each 50,000 total number of pages. ACEBot achieves nearly 80% of the proportion of seen *n-grams* with 2,000 sample pages, though stable performance for different Web sites is achieved over 2,500 sample pages. Since the sample pages are chosen randomly, and a set of navigation patterns learned for several runs may vary, we have run ACEBot for each sample size (e.g., 500) 10 times, to evaluate the performance over several run. For sample size 500, the more inconsistent navigation patterns are selected. ACEBot has seen 6 times (consider the plot point label for size 500) the same amount of distinct *n-grams* with size 500, whereas with size 3,000, every time the same navigation patterns were selected. Clearly, site map construction with size 3,000 yields the best result and performs consistently.

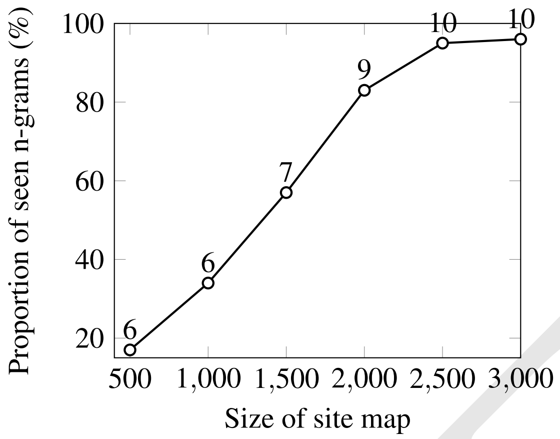


Figure 6: Proportion of seen *n-grams* for 10 Web sites as the number of sample pages varies; plot labels indicate the number of times, out of 10 runs on the same Web sites, the maximum number of distinct n-grams was reached

**Algorithm** We will consider several settings for our proposed algorithm 1. The additional parameters *d*, *cr*, *k*, *ac*, and *a* form several variants of our technique: The sitemap *d* may be dynamic (limiting to 3,000 Web pages; default if not otherwise specified) or complete (whole Web site mirror). The completion ration *cr* may take values 85%, 90%, 95% (default). The level depth *k* is set to either 2, 3 (default), or 4. The continuity limit *ac*, that specifies whether to limit repetition of navigation patterns to a fixed number (3, default) or consider arbitrary number of repetitions, is a Boolean. Finally, the attributes used, *a*, may be set to *id* (default), *class*, or both. Our system was tested for all these variants to evaluate the performance for different settings. We describe next the results, that we found highly stable and consistent. Occasionally, we present experiments for values of the parameters beyond the ones above-mentioned.

**Performance Metrics** We have compared the performance of ACEBot with AAH and GNU wget, by evaluating the number of HTTP requests made by these crawlers vs the number of useful content retrieved. We have considered the same performance metrics used by AAH [10], where the evaluation of number of HTTP requests is performed by simply counting the requests. Coverage of useful content is evaluated by comparing the proportion of 2-grams (sequences of two consecutive words) in the crawl result of three systems, for every Web site, and by counting the number of external links (i.e., hyperlinks to another domain) found in the three crawls. External links are considered an important part of the content of a Web site.

## 4.2 Crawl Efficiency

We have computed the number of pages crawled with ACEBot, AAH, and GNU wget, to compare the crawl efficiency of the three systems (see Figure 7). Here, wget obviously crawls 100% of the dataset. ACEBot makes 5 times fewer requests than blind crawl, and slightly more than AAH, the latter being only usable for the three CMS it handles. The performance of ACEBot remains stable when we experimented with WordPress, vBulletin, phpBB, and with other template-based

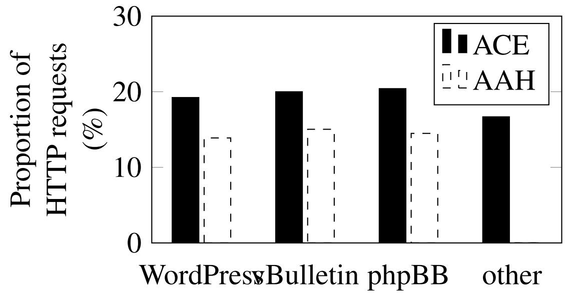


Figure 7: Total number of HTTP requests used to crawl the dataset, in proportion to the total size of the dataset, by type of Web site

Web sites. ACEBot exploits the link structure of a given Web site, and selects a set of navigation patterns with high score (see Definition 4). Therefore, our approach is not much affected by noisy links, invalid, and duplicates pages. Indeed, our approach avoids redundant requests for the same Web content (e.g., access a blog post by different services: tag, year, author, print view). Since all redundant pages have different navigation patterns, if one navigation pattern is selected then the other navigation patterns (leading to the redundant pages) will have lower score and hence will be automatically discarded.

The results shown in Figure 8 plot the number of seen 2-grams, and the number of HTTP requests made for a selected number of navigation patterns. The number of HTTP requests and discovered *n-grams* for a navigation pattern decide of its selection (a navigation pattern with best score is selected). Therefore a navigation patterns with one single page, but with many new *n-grams* may be selected ahead of a navigation pattern with many HTTP requests. Indeed, the Figure 8 elaborates that prospect, where the 10th selected navigation pattern crawl a large number of pages but this navigation pattern was selected only because it reached a higher completion ratio. The higher completion ratio ensure that the optimal number of navigation patterns are selected, including a navigation pattern with most (but important) number of HTTP requests.

## 4.3 Crawl Effectiveness

ACEBot crawling results in terms of coverage of useful content are summarized in Figures 9, 10, 11, and in Table 2.

Figure 9 illustrates the proportion of crawled *n-grams* by ACEBot for three possible combination of attributes *id*, and *class* with both complete (whole mirror of Web site) and dynamic (randomly selected 3000 pages) site maps. The experiments depicts the importance of a specific attribute by comparing the coverage of useful content. Figures 9a, 9b exhibit the results for a complete site map, whereas Figures 9c, 9d show the effectiveness for a dynamic sitemap. Further, a set of navigation patterns are learned by limiting (or not) the continuity constraints (for the action of + type). The experiments has shown that the navigation patterns with attribute *class* has less significance than *id* or when we consider the both attributes (i.e., *id*, and *class*). The *class* attribute may have multiple values, and several *css* style may be given to a node. Therefore, one has to consider that we ignore the multiple values for class attribute in navigation pattern learning phase and just add a constraint `[@class]` for the node (e.g., `/div[@class]`). When we

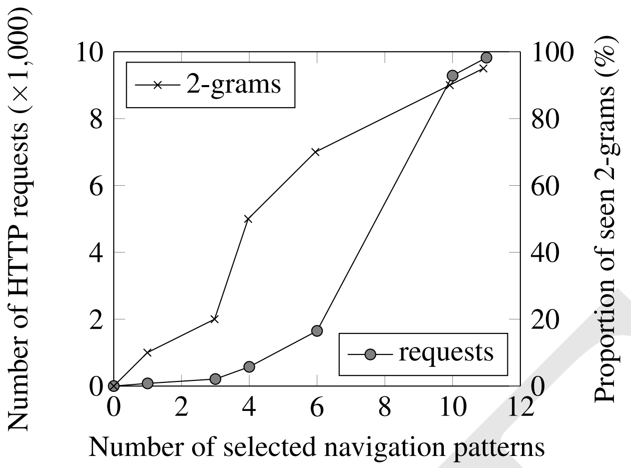


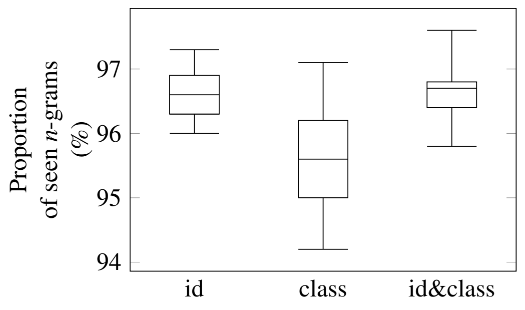
Figure 8: Number of HTTP requests and proportion of seen n-grams for 10 Web sites as the number of selected navigation patterns increase

consider both attributes for learning the navigation patterns, the performance is more stable, but importantly, the learned navigation may work for the specific Web site and may lack the coverage of useful content when executed for similar kinds of Web sites. The *id* attribute has achieved a similar result, except for few cases, when higher coverage was achieved with both attributes. Also, it is to be expected that selected navigation patterns comprising just *id* attributes may show better performance for reuse in similar Web sites. ACEBot has achieved over 96% (median) effectiveness with both complete and dynamic sitemaps and whether (or not) the continuity action is restricted. The results are also very stable with very low variance: the worst coverage score for our whole dataset is 96% for complete site map and over 95% for dynamic site map with both attributes. Moreover, the restriction on action continuity does not effect the performance. This indeed show the statistical significance of our results, and the appropriateness of limiting the continuity action to 3 repetitions.

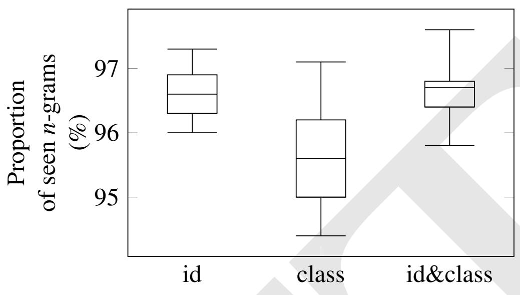
The proportion of coverage of useful content and external links for different navigation steps (level) is shown in Table 2. Limiting navigation patterns to level 2 or 3 results in less HTTP requests, and a performance of 96% content with 95% completion ratio. But the level 3 performs better across many Web sites in terms of effectiveness, as important content exist till link depth 3. Once the learned navigation patterns achieve the 95% coverage of *n-grams* vs whole blind crawl, the selected navigation patterns will be stored in a knowledge base for future re-crawling.

Figure 10 evaluate the performance of ACEBot for different completion ratio for 10 selected Web site, each with 50,000 Web pages. The selection of a set of navigation patterns ends when the completion ratio has been achieved. The experiments have shown that the higher (and stable) proportion of *n-grams* are seen with completion ratio over 80%.

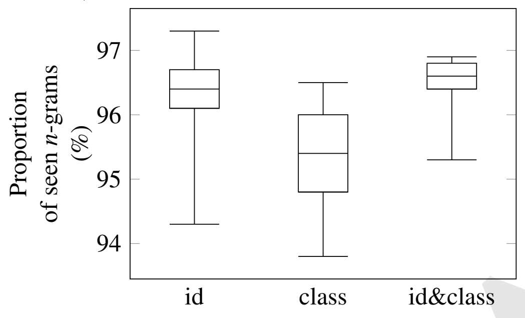
The proportion of external links coverage by ACEBot is given in Table 2. Since ACEBot selects the best navigation patterns and achieves higher content coverage, over 99% external links are present in the content crawled by ACEBot for the whole dataset.



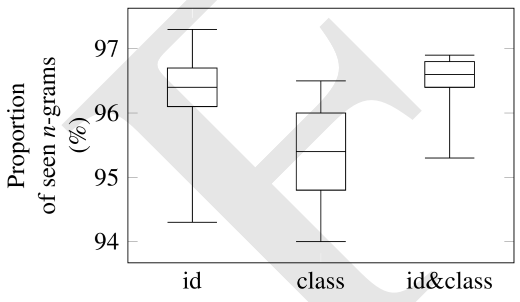
(a) Complete sitemap (with no restriction on + actions).



(b) Complete sitemap (with restriction on +).



(c) Dynamic sitemap (with no restriction on +).



(d) Dynamic sitemap (with restriction on +).

Figure 9: Box chart of the proportion of seen *n*-grams in the whole dataset for different settings: minimum and maximum values (whiskers), first and third quartiles (box), median (horizontal rule)

## 4.4 Comparison to AAH

To reach a better understanding of the performance of ACEBot, we plot in Figure 11, the number of distinct *2-grams* seen by ACEBot, AAH, and wget during one crawl, as the number of requests increase. Clearly, AAH [10], and ACEBot directly crawl the interesting content of the Web site and newly discovered *2-grams* grow linearly with number of requests made. The results show the performance of automatic structure-based crawler (ACEBot) as close to the semi-automatic crawler (AAH). ACEBot makes 2,448 requests to crawl 97% of *2-grams* coverage, as compared to 92% content coverage with 2,200 requests by AAH. Since AAH is based on hand-written knowledge base, the engineer may miss to specify the navigation patterns to crawl other important content (e.g., interview pages in the case at hand). This indeed illustrates that an automatic approach not only crawls important pages, but also ensure that all portion of Web site has considered.

The experiments of AAH [10] are performed for 100 Web sites (nearly 3.3 million Web pages). To compare ACEBot to AAH more globally, we have crawled 10 of the same Web sites (nearly 0.5 million Web pages) used in AAH [10]. ACEBot is fully automatic, whereas the AAH has introduces a semi-automatic approach (still domain dependent) and thus requires a hand-written knowledge base (XML format) to initiate a bulk downloading of known Web application. Over 96 percent crawl effectiveness in terms of *2-grams*, and over 99 percent in terms of external links

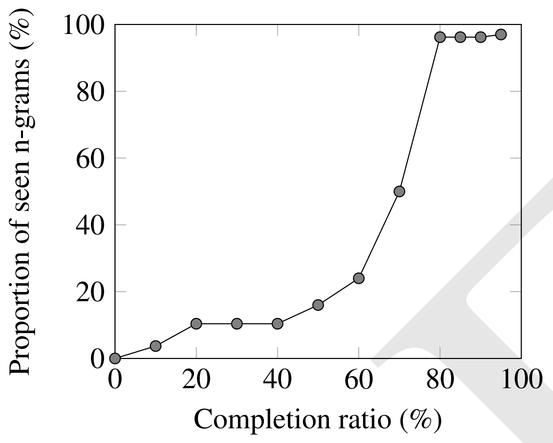


Figure 10: Proportion of seen *n-grams* for different completion ratios for 10 Web sites

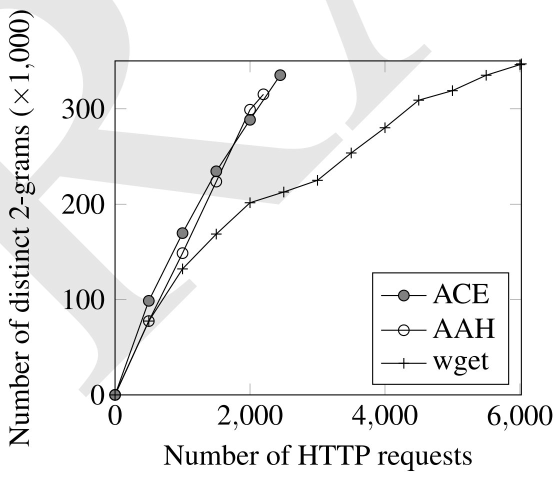


Figure 11: Crawling `http://www.rockamring-blog.de/`

| Level | Requests | Content (%) | External Links (%) | Completion ratio (%) |
|---|---|---|---|---|
| 2 | 376632 | 95.7 | 98.6 | 85 |
|  | 377147 | 95.8 | 98.7 | 90 |
|  | 394235 | 96.0 | 99.1 | 95 |
| 3 | 418654 | 96.3 | 99.2 | 85 |
|  | 431572 | 96.6 | 99.3 | 90 |
|  | 458547 | 96.8 | 99.3 | 95 |
| 4 | 491568 | 96.9 | 99.4 | 85 |
|  | 532358 | 97.1 | 99.4 | 90 |
|  | 588512 | 97.2 | 99.4 | 95 |

Table 2: Performance of ACEBot for different levels with dynamic sitemap (restriction on +) for the whole data set (2 million pages)

is achieved for ACEBot, as compared to over 99 percent content completeness (in terms of both *2-grams* and external links) for AAH for the same Web sites. The lower content retrieval for ACEBot than for AAH is naturally explained by the 95% target completion ration considered for ACEBot. However it is also important to note that the performance of AAH relies on the hand written crawling strategy described in knowledge base by a crawl engineer. The crawl engineer must be fully aware to the Web pages structure (as well as the link organization) for the crawled Web site, to effectively download the important portion, as contrasted to our fully automatic approach, where one does not need to know such information for effective downloading and automatically learn the important portion of Web site. The current approach makes 5 times fewer HTTP requests as compared to 7 times for AAH (See Figure 7). Indeed the current effort crawls more pages than the AAH, but is fully unsupervised.

## 4.5 Comparison to iRobot

We have performed the comparison of our approach with the iRobot system [5]. iRobot is not available for testing because of intellectual property reasons. The experiments of [5] are performed just for 50,000 Web pages, over 10 different forum Web sites (to compare with our evaluation, on 2.0 million Web pages, over 50 different Web sites). To compare ACEBot to iRobot, we have crawled one of the same Web forum used in [5]: `http://www.tripadvisor.com/ForumHome` (over 50,000 Web pages). The completeness of content of the our system is nearly 97 percent in terms of *2-grams*, and 100 percent in terms of external links coverage; iRobot has a coverage of *valuable content* (as evaluated by a human being) of 93 percent on the same Web site. The crawl efficiency (in terms of number of HTTP requests) for iRobot is claimed in [5] to be 1.73 times less than a regular Web crawler; on the `http://www.tripadvisor.com/ForumHome` Web application, ACEBot makes 5 times fewer requests than wget does.

## 5 Related work

Several approaches have been proposed in the literature to implement an efficient system for Web crawling, we review them here. Note that we do not discuss here *focused crawling* [6], which is the different goal of crawling with respect to a given topic or intent, see [10] for a comparison between focused crawling and efficient crawling of Web applications.

In [16], Liu et al. proposed an algorithm, called *SEW*, that models a Web site as a hypertext structure. The skeleton is organized in a hierarchical manner, with nodes either navigation pages or content pages. The navigation pages contain links to the content pages, whereas the content pages provide the information content. SEW relies on a combination of several domain-independent heuristics to identify the most important links within a Web page and thus discover a hierarchical organization of navigation and content pages. Kao et al. [13] have addressed a similar problem, and they propose a technique to distinguish between pages containing *links* to news posts and the pages containing these news. Their proposed method eliminates the redundancy of hypertext structure using entropy-based analysis; similarly a companion algorithm is used to discard redundant information. Compared to our approach, the both techniques above only cluster the Web pages into two predefined classes of pages: navigational and content pages. In addition, Kao et al. [13] focus on pages of a specific domain (news). In contrast, we have proposed a system that performs unsupervised crawling of Web sites (domain independent). Our system may have several classes and, indeed, the crawler will follow the best traversal path to crawl the content-rich area. In addition, our approach does not make any prior assumption about the number of classes.

In [7, 8, 2], a similar version to our model has been adopted. [7, 8] aim to cluster Web pages into different classes by exploiting their structural similarity at the DOM tree level, while [2] introduces crawling programs: a tool (implemented as a Firefox plugin) that listen to the user interaction, registers steps, and infers the corresponding intentional navigation. A user is just required to browse the Web site towards a page of interest of a target class. This approach is semi-supervised as it requires human interaction to learn navigation patterns to reach the content-rich pages. A Web crawler is generally intended in a massive crawling scenario, and thus semi-automatic approaches would require a lot of human interaction for each seed Web site, not feasible in our setting. Therefore, in our proposed approach, we have introduced the offline phase which learns navigation patterns (that leads to content-rich pages) in an unsupervised manner and hence will have better Web-scale performance.

Another structure-driven approach [23, 22], has proposed a Web crawler, named *GoGetIt!* [22] that requires minimum human effort. It takes a sample page (page of interest) and entry point as input and generates a set of navigation patterns (i.e., sequence of patterns) that guides a crawler to reach Web pages structurally similar to the sample page. As stated above, this approach is also focused on a specific type of Web page, whereas our approach performs massive crawling at Web scale for content-rich pages. Similarly, [14] presents a recipe crawler (domain dependent) that uses certain features to retrieve the recipe pages.

Several domain-dependent *Web forum* crawling techniques [12, 5, 15] have been proposed recently. In [12], the crawler first clusters the Web pages into two groups from a set of manual annotated pages using Support Vector Machines with some predefined features, and then, within each cluster, URLs are clustered using partial tree alignment. Further a set of ITF (index-threadpage-flipping) *regexes* are generated to launch a bulk download of a target Web forum. The iRobot system [5], that we use as a baseline in our experiments, creates a sitemap of the Web site being crawled. The sitemap is constructed by randomly crawling a few Web pages from a given Web site. After sitemap generation, iRobot obtains the structure of the Web forum. The skeleton is obtained in the form of a directed graph consisting of vertices (Web pages) and directed arcs (links between different Web pages). Furthermore, a path analysis is performed to learn an optimal traversal path which leads the extraction process in order to avoid duplicate and invalid page. A lightweight algorithm [15] performs content based features to cluster the links for an automatic crawling of Web forums. This approach first identify the signature and common keywords for the reference links on the forum posts. Then it finds XPath expressions for each content page and content regions within it. An extraction phase is introduced to fetch the actual content. This approach also does not perform on Web scale but restricted to specific Web forums.

A Web scale approach [3] has introduced an algorithm that performs URL-based clustering of Web pages using some content features. However, in practice, URL-based clustering of Web pages is less reliable in the presence of dynamic nature of Web. Our previous work [10] proposes an adaptive application-aware helper (AAH) that crawl known Web sites efficiently. AAH is assisted with a knowledge base (specifies the Web sites detection parameters and relevant crawling actions) that guides the crawling process. It first tries to detect the Web site and, if a Web site is detected as a known one, attempts to identify the kind of Web page given the matched Web site. Once the kind of Web page has been matched, the relevant crawling actions are executed for Web archiving. This approach achieves the highest quality of Web content with fewer HTTP requests, but is not fully automatic and requires a hand-written knowledge base that prevents crawling of unknown Web sites. The AAH is part of the ARCOMEM crawling architecture [20], where the purpose of this module is to make a large-scale crawler aware of the type of Web site currently processed, refines the list of URLs to process, and extracts structured information from crawled Web pages.

## 6 Conclusions

In this paper, we have introduced an Adaptive Crawler Bot for data Extraction (ACEBot), that utilizes the inner structure of Web pages, rather than their content or URL-based clustering techniques, to determine which pages are important to crawl. Extensive experiments over a large dataset has shown that our proposed system performs well for Web sites that are data-intensive and, at the same time, present regular structure. We have compared the performance of ACEBot with generic crawler GNU wget, AAH, and iRobot. Our system significantly reduces duplicate, noisy links, and invalid pages without compromising on coverage of useful content, achieving high crawl quality as well. ACEBot only require 3,000 sample pages to construct a site map. Compared with generic crawler, ACEBot makes 5 times fewer HTTP requests, slightly more than AAH (but AAH only handle three CMSs and also require hand-written knowledge base). ACEBot has also outperformed iRobot, that achieves 93% crawl effectiveness, compared to 97% in terms of useful content and 100% in terms of external links with ACEBot.

## References

[1] J. Alpert and N. Hajaj. We knew the web was big... `http://googleblog.blogspot.co.uk/2008/07/we-knew-web-was-big.html`, 2008.

[2] C. Bertoli, V. Crescenzi, and P. Merialdo. Crawling programs for wrapper-based applications. In *IRI*, 2008.

[3] L. Blanco, N. N. Dalvi, and A. Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *WWW*, 2011.

[4] G. Brumfiel. The first web page, amazingly, is lost. `http://www.npr.org/2013/05/22/185788651/the-first-web-page-amazingly-is-lost`, 2013.

[5] R. Cai, J.-M. Yang, W. Lai, Y. Wang, and L. Zhang. iRobot: An intelligent crawler for Web forums. In *WWW*, 2008.

[6] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. In *WWW*, 1999.

[7] V. Crescenzi, P. Merialdo, and P. Missier. Fine-grain Web site structure discovery. In *WIDM*, 2003.

[8] V. Crescenzi, P. Merialdo, and P. Missier. Clustering Web pages based on their structure. *Data Knowl. Eng.*, 54(3):279–299, 2005.

[9] M. de Kunder. The indexed Web. `http://www.worldwidewebsize.com/`, 2013.

[10] M. Faheem and P. Senellart. Intelligent and adaptive crawling of Web applications for Web archiving. In *Proc. ICWE*, 2013.

[11] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of Web page templates. In *WWW*, 2005.

[12] J. Jiang, X. Song, N. Yu, and C.-Y. Lin. Focus: Learning to crawl Web forums. *IEEE Trans. Knowl. Data Eng.*, 2013.

[13] H.-Y. Kao, S.-H. Lin, J.-M. Ho, and M.-S. Chen. Moining Web informative structures and contents based on entropy analysis. *IEEE Trans. Knowl. Data Eng.*, 2004.

[14] Y. Li, X. Meng, L. Wang, and Q. Li. RecipeCrawler: Collecting recipe data from WWW incrementally. In *Advances in Web-Age Information Management*. Springer, 2006.

[15] W.-Y. Lim, A. Sachan, and V. L. L. Thing. A lightweight algorithm for automated forum information processing. In *Web Intelligence*, 2013.

[16] Z. Liu, W. K. Ng, and E.-P. Lim. An automated algorithm for extracting Website skeleton. In *DASFAA*, 2004.

[17] J. Masanès. *Web archiving*. Springer, 2006.

[18] F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz. Evaluating topic-driven web crawlers. In *SIGIR*, 2001.

[19] X. Ochoa and E. Duval. Quantitative analysis of user-generated content on the Web. In *WebEvolve*, 2008.

[20] V. Plachouras, F. Carpentier, M. Faheem, J. Masanés, T. Risse, P. Senellart, P. Siehndel, and Y. Stavrakas. ARCOMEM crawling architecture. *Future Internet*, 6, 2014.

[21] A. Sellers, T. Furche, G. Gottlob, G. Grasso, and C. Schallhart. Exploring the Web with OXPath. In *LWDM*, 2011.

[22] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Cavalcanti. GoGetIt!: a tool for generating structure-driven Web crawlers. In *WWW*, 2006.

[23] M. L. A. Vidal, A. S. da Silva, E. S. de Moura, and J. M. B. Cavalcanti. Structure-driven crawler generation by example. In *SIGIR*, 2006.

[24] W3C. Document Object Model (DOM) Level 1 specification, W3C Recommendation. `http://www.w3.org/TR/REC-DOM-Level-1`, 1998.