Intelligent and Adaptive Crawling of Web Applications for Web Archiving

Muhammad Faheem[1] and Pierre Senellart[1,2]

[1] Institut Mines–Télécom, Télécom ParisTech, CNRS LTCI, Paris, France
[2] The University of Hong Kong, Hong Kong
firstname.lastname@telecom.paristech.fr

Abstract. Web sites are dynamic in nature with content and structure changing overtime. Many pages on the Web are produced by content management systems (CMSs) such as WordPress, vBulletin, or phpBB. Tools currently used by Web archivists to preserve the content of the Web blindly crawl and store Web pages, disregarding the CMS the site is based on (leading to suboptimal crawling strategies) and whatever structured content is contained in Web pages (resulting in page-level archives whose content is hard to exploit). We present in this paper an *application-aware helper* (AAH) that fits into an archiving crawl processing chain to perform intelligent and adaptive crawling of Web applications (e.g., the pages served by a CMS). Because the AAH is aware of the Web application currently crawled, it is able to refine the list of URLs to process and to extend the archive with semantic information about extracted content. To deal with possible changes in structure of Web applications, our AAH includes an adaptation module that makes crawling resilient to small changes in the structure of Web site. We show the value of our approach by comparing the output and efficiency of the AAH with respect to regular Web crawlers, also in the presence of structure change.

1 Introduction

Social Web Archiving. The World Wide Web has become an active publishing system and is a rich source of information, thanks to contributions of hundreds of millions of Web users. Part of this public expression is carried out on social networking and social sharing sites (Twitter, Facebook, Youtube, etc.), part of it on independent Web sites powered by content management systems (CMSs, including blogs, wikis, news sites with comment systems, Web forums). Content published on this range of *Web applications* includes information that is newsworthy today or valuable to tomorrow's historians. Barack Obama thus first announced his 2012 reelection as US president on Twitter [1]; blogs are more and more used by politicians both to advertise their political platform and to listen to citizens' feedback [2]; Web forums have become a common way for political dissidents to discuss their agenda [3]; user-contributed wikis such as Wikipedia contain quality information to the level of traditional reference materials [4].

F. Daniel, P. Dolog, and Q. Li (Eds.): ICWE 2013, LNCS 7977, pp. 306–322, 2013.

Because Web content is distributed, perpetually changing, often stored in proprietary platforms without any long-term access guarantee, it is critical to preserve this valuable material for historians, journalists, or social scientists of future generations. This is the objective of *Web archiving* [5], which deals with discovering, crawling, storing, and ensuring long-term access to Web data.

Application-Aware Archiving. Current archival crawlers, such as Internet Archive's Heritrix [6], function in a conceptually simple manner. They start from a *seed* list of URLs to be stored in a queue. Web pages are then fetched from this queue one after the other (respecting crawling ethics, limiting the number of requests per server), stored as is, and links are extracted from them. If these links point to resources in the scope of the archiving task, they are added to the queue. This process ends after a specified time or when no new relevant URL can be found.

This approach does not confront the challenges of modern Web application crawling: the nature of the Web application crawled is not taken into account to decide the crawling strategy or the content to be stored; Web applications with dynamic content (e.g., Web forums, blogs, etc.) may be crawled inefficiently, in terms of the number of HTTP requests required to archive a given site; content stored in the archive may be redundant, and typically does not have any structure (it consists of flat HTML files), which makes access to the archive cumbersome.

The aim of this work is to address this challenge by introducing a new *application-aware* approach to archival Web crawling. Our system, the *application-aware helper* (AAH for short) relies on a knowledge base of known Web applications. A *Web application* is any HTTP-based application that utilizes the Web and Web browser technologies to publish information using a specific template. We focus in particular on social aspects of the Web, which are heavily based on user-generated content, social interaction, and networking, as can be found for instance in Web forums, blogs, or on social networking sites. Our proposed AAH only harvests the important content of a Web application (i.e., the content that will be valuable in a Web archive) and avoids duplicates, uninteresting URLs and templates that just serve a presentational purpose. In addition the application-aware helper extracts from Web pages individual items of information (such as blog post content, author, timestamp).

To illustrate, consider the example of a Web forum, say, powered by a content management system such as vBulletin. On the server side, forum threads and posts are stored in a database; when a user requests a given Web page, the response page is automatically generated from this database content, using a predefined template. Frequently, access to two different URLs will end up presenting the same or overlapping content. For instance, a given user's posts can be accessed both through the classical threaded view of forum posts or through the list of all his or her post displayed on the user profile. This redundancy means that an archive built by a classical Web crawler will contain duplicated information, and that many requests to the server do not result in novel pieces of content. In extreme cases, the crawler can fall into a *spider trap* because it has infinitely many links to crawl. There are also several noisy links such as to a

print-friendly page or advertisement, etc., which would be better to avoid during the constitution of the archive. On the contrary, a Web crawler that is aware of the information to be crawled can determine an optimal path to crawl all posts of a forum, without any useless requests, and can store individual posts, together with their authors and timestamps, in a structured form that archivists and archive users can benefit of.

Template Change. Web applications are dynamic in nature; not only their content changes over time, but their structure and template does as well. Content management systems provide several templates that one can use for generating wiki articles, blog posts, forum messages, etc. These systems usually provide a way for changing the template without altering the informational content, to adapt to the requirements of a specific site. The layout may also change as a new version of the CMS is installed. All these layout changes result in possible changes in the DOM tree of the Web page, usually minor. This makes it more challenging to recognize and process in an intelligent manner all instances of a given content management systems, as it is hopeless to hope to manually describe all possible variations of the template in a Web application knowledge base. Another goal of this work is an intelligent crawling approach that is resilient to minor template changes, and, especially, automatically adapts to these changes, updating its knowledge of CMSs in the process. Our adaptation technique relies on both relaxing the crawling and extraction patterns present in the knowledge base, and on comparing successive versions of the same Web page.

Outline. After presenting the related work (Sect. 2) and giving some preliminary definitions (Sect. 3), we describe our knowledge base of Web applications in Sect. 4. The methodology that our application-aware helper implements is then presented in Sect. 5. We discuss the specific problem of adaptation to template changes in Sect. 6 before covering implementation issues and explaining how the AAH fits into a crawl processing chain in Sect. 7. We finally compare the efficiency and effectiveness of our AAH with respect to classical crawling approach in crawling blogs and Web forums in Sect. 8. Initial ideas leading to this work were presented as a PhD workshop article in [7]; the description of the algorithms and system, adaptation to template change, experimental results, are fully novel.

2 Related Work

Web Crawling. Web crawling is a well-studied problem with still ongoing challenges. A survey of the field of Web archiving and archival Web crawling is available in [5]. A *focused*, or *goal-directed*, crawler, crawls the Web according to a predefined set of topics [8], and thus influences the crawler behavior not based on the structure of Web applications as is our aim, but on the content of Web pages. Our approach does not have the same purpose as focused crawling: it aims at better archiving of known Web applications. Both strategies for are thus complementary.

Content in Web applications or *content management systems* is arranged with respect to a template (which may include left or right sidebar of the Web page, navigation bar, header and footer, main content, etc.). Among the various works on template extraction, Gibson et al. [9] have performed an analysis of the extent of template-based content on the Web. They have found that 40–50% of the Web content (in 2005) is template-based (i.e., part of some Web application), which is growing at the rate of 6–8% per year. This research is a strong hint at the benefit of handling Web application crawling in a specific manner.

Forum Crawling. Though application-aware crawling in general has not yet been addressed, there have been some efforts on content extraction from Web forums. One such approach [10], dubbed Board Forum Crawling (BFC), leverages the organized structure of Web forums and simulates user behavior in the extraction process. BFC deals with the problem effectively, but is still confronted to limitations as it is based on simple rules and can only deal with forums with some specific organized structure.

Another technique [11], however, does not depend on the specific structure of the Web forum. The iRobot system assists the extraction process by providing the sitemap of the Web application being crawled. The sitemap is constructed by randomly crawling a few pages from the Web application. After sitemap generation, iRobot obtains the structure of the Web forum 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 provide an optimal traversal path which leads the extraction process in order to avoid duplicate and invalid pages. A later effort [12] identified a few drawbacks in iRobot and improved the original system in a number of way: a better minimum spanning tree discovery technique [13], a better measure of the cost of an edge in the crawling process as an estimation of its approximate depth in the site, and a refinement of the detection of duplicate pages. iRobot [11,12] is probably the work closest to ours. In contrast with that system, the AAH we propose is applicable to any kind of Web application, as long as it is described in our knowledge base. Also differently from [11,12], where the analysis of the structure of a forum has to be done independently for each site, the AAH exploits the fact that several sites may share the same content management system. Our system also extracts structured and semantic information from the Web pages, where iRobot stores plain HTML files and leaves the extraction for future work. We finally give in Sect. 8 a comparison of the performance of iRobot vs AAH to highlight the superior efficiency of our approach. On the other hand, iRobot aims at a fully automatic means of crawling a Web forum, while the AAH relies on a knowledge base (manually constructed but automatically maintained) of known Web applications or content management systems.

Web Application Detection. As we shall explain, our approach relies on a generic mechanism for detecting the kind of Web application currently crawled. Again there has been some work in the particular cases of blogs or forums. In particular, [14] uses support vector machines (SVM) to detect whether a given page is a

blog page. In [14], SVMs are trained using various traditional feature vectors formed of the content's bag of words or bag of n-grams, and some new features for blog detection are introduced such as the bag of linked URLs and the bag of anchors. Relative entropy is used for feature selection.

Wrapper Adaptation. Wrapper adaptation, the problem of adapting a Web information extractor to (minor) changes in the structure of considered Web pages or Web sites, has received quite some attention in the research community. An early work is that of Kushmerick [15] who proposed an approach to analyze Web pages and already extracted information, so as to detect changes in structure. A "wrapper verification" method is introduced that checks whether a wrapper stops extracting data; if so, a human supervisor is notified so as to retrain the wrapper. Chidlovskii [16] introduced some grammatical and logic-based rules to automate the maintenance of wrappers, assuming only slight changes in the structure of Web pages. Meng, Hu, and Li [17] suggested a schema-guided wrapper maintenance approach called SG-WRAM for wrapper adaptation. Lerman, Minton, and Knoblock [18] developed a machine learning system for repairing wrapper for small markup changes. Their proposed system first verifies the extraction from Web pages, and if the extraction fails then it relaunches the wrapper induction for data extraction.

Our template adaptation technique is inspired by the previously cited works: we check whether patterns of our wrapper fail, and if so, we try fixing them assuming minor changes in Web pages, and possibly using previously crawled content on this site. One main difference with existing work is that our approach is also applicable to completely new Web sites, never crawled before, that just share the same content management system and a similar template.

Data Extraction from Blogs, Forums, etc. A number of works [19,20,21] aim at automatic wrapper extraction from CMS-generated Web pages, looking for repeated structure and typically using tree alignment or tree matching techniques. This is out of scope of our approach, where we assume that we have a preexisting knowledge base of Web applications. Gulhane et al. [22] introduced the Vertex wrapper induction system. Vertex detects site changes by monitoring a few sample pages per site. Any structural change can result in changes in page

```
⟨expr⟩       ::= ⟨step⟩ | ⟨step⟩ "/"  ⟨expr⟩
                         ⟨step⟩ "//" ⟨expr⟩
⟨step⟩       ::= ⟨nodetest⟩ | ⟨step⟩ "[" ⟨predicate⟩ "]"
⟨nodetest⟩   ::= tag | "@" tag | "*" | "@*" | "text()"
⟨predicate⟩  ::= "contains(" ⟨value⟩ "," string ")" |
                 ⟨value⟩ "=" string | integer | "last()"
⟨value⟩      ::= tag | "@" tag
```

Fig. 1. BNF syntax of the XPath fragment used. The following tokens are used: *tag* is a valid XML identifier; *string* is a single- or double-quote encoded XPath character string; *integer* is any positive integer.

shingle vectors that may render learned rules inapplicable. In our system, we do not monitor sampled Web pages but dynamically adapt to pages as we crawl them. The AAH also applies adaptation to different versions of a content management system found on different Web sites, rather than to just a specific Web page type.

3 Preliminaries

This section introduce some definitions that we will use throughout this paper. A *Web application* is any application or Web site that uses Web standards such as HTML and HTTP to publish information on the Web in a specific template, in a way that is accessible by Web browsers. Examples include Web forums, social networking sites, geolocation services, etc. A *Web application type* is the content-management system or server-side technology stack (e.g., vBulletin, WordPress, the proprietary CMS of Flickr, etc.) that powers this Web application and provides interaction with it. Several different Web applications can share the same Web application type (all vBulletin forums use vBulletin), but some Web application types can be specific to a given Web application (e.g., the CMS powering Twitter is specific to that site).

We use a simple subset of the XPath expression language to describe *patterns* in the DOM of Web pages that serve either to identify a Web application type, or to determine navigation or extraction *actions* to apply to that Web page. A grammar for the subset we consider is given in Fig. 1. Basically, we only allow downwards axes and very simple predicates that perform string comparisons. The semantics of these expressions is the standard one. In the following, an *XPath expression* is always one of this sublanguage.

A *detection pattern* is a rule for detecting Web application types and Web application, based on the content of a Web page, HTTP metadata, URL components. It is implemented as an XPath expression over a virtual document that contains the HTML Web page as well as all other HTTP metadata.

A *crawling action* is an XPath expression over an HTML document that indicates which action to perform on a given Web page. Crawling actions can be of two kinds: *navigation actions* point to URLs to be added to the crawling queue; *extraction actions* point to individual semantic objects to be extracted from the Web page (e.g., timestamp, blog post, comment). For instance, `div[contains(@class,'post')]//h2[@class='post-message']//a/@href` is a navigation action to follow certain types of links.

The application-aware helper distinguishes two main kinds of Web application *levels*: *intermediate* pages, such as lists of forums, lists of threads, can only be associated with navigation actions; *terminal pages*, such as the individual posts in a forum thread, can be associated with both navigation and extraction actions. For intelligent crawling, our AAH needs not only to distinguish among Web application types, but among the different kinds of Web pages that can be produced by a given Web application type. The idea is that the crawler will navigate intermediate pages until a terminal page is found, and only content

from this terminal page is extracted; the terminal page may also be navigated, e.g., in the presence of paging.

Given an XPath expression e, a *relaxed expression* for e is one where one or several of the following transformations has been performed:

- a predicate has been removed;
- a *tag* or *string* token has been replaced with another such token.

A *best-case relaxed expression* for e is one where at most one of these transformations has been performed for every step of e. A *worst-case relaxed expression* for e is one where potentially multiple transformations have been performed on any given step of e.

To illustrate, consider e =`div[contains(@class,'post')]//h2[@class='post-message']`. Examples of best-case relaxed expressions are `div[contains(@class,'post')]//h2` or `div[contains(@class,'post')]//h2[@id='post-content']`; on the other hand, `div[contains(@class,'message')]//div[@id='post-content']` is an example worst-case relaxed expression.

4 Knowledge Base

The AAH is assisted by a *knowledge base* of Web application types. This knowledge base specifies how to detect specific Web applications and which crawling actions should be executed. Types are arranged in a hierarchical manner, from general categorizations to specific instances (Web sites) of this Web application. The knowledge base also describes the different levels under a Web application type and then, based on this, different crawling actions that should be executed against this specific page level. The knowledge base is specified in a declarative language, so as to be easily shared and updated, hopefully maintained by non-programmers, and also possibly automatically learned from examples. The W3C has normalized a Web Application Description Language (WADL) [23] for describing resources of HTTP-based applications. WADL does not satisfy all our needs: in particular, there is no place for the description of Web application recognition patterns. Consequently, our knowledge-based is described in a custom XML format.

For each Web application type, and for each level, the knowledge base contains a set of detection patterns that allows to recognize whether a given page is of that type or that level. The vBulletin Web forum CMS can for instance be identified by searching for a reference to a specific script with the detection pattern: `script[contains(@src,'vbulletin_global.js')]`. Pages of the "list of forums" type are identified[1] when they match the pattern `a[@class="forum"]/@href`.

Similarly, for each Web application type and level, a set of navigation and extraction actions (for the latter, only in the case of terminal levels) is provided.

[1] The example is simplified for the sake of presentation; in reality we have to deal with several different layouts that vBulletin can produce.

5 Application-Aware Helper (AAH)

Our main claim is that different crawling techniques should be applied to different types of Web applications. This means having different crawling strategies for different forms of social Web sites (blogs, wikis, social networks, social bookmarks, microblogs, music networks, Web forums, photo networks, video networks, etc.), for specific content management systems (e.g., WordPress, phpBB), and for specific sites (e.g., Twitter, Facebook). Our proposed approach will detect the type of Web application (general type, content management system, or site) currently processed by the crawler, and the kind of Web pages inside this Web application (e.g., a user profile on a social network) and decide on further crawling actions (following a link, extracting structured content) accordingly. The proposed crawler is intelligent enough to crawl and store all comments related to a given blog post in one place, even if comments stays on several Web pages.

The AAH detects the Web application and Web page type before deciding which crawling strategy is appropriate for the given Web application. More precisely, the AAH works in the following order:

1. it detects the Web application type;
2. it detects the Web application level;
3. it executes the relevant crawling actions: extracting the outcome of extraction actions, and adding the outcome of navigation actions to the URL queue.

The AAH loads the Web application type detection patterns from the knowledge base and executes them against the given Web application. If the Web application type is detected, the system executes all the possible Web application level detection patterns until it gets a match.

The number of detection patterns for detecting Web application type and level will grow with the addition of knowledge about new Web applications. In order to optimize this detection, the system needs to maintain an index of these patterns. To this aim, we have integrated the YFilter system [24] (an NFA-based filtering system for XPath expressions) with slight changes according to our requirements, for efficient indexing of detection patterns, in order to quickly find the relevant Web application types and levels. YFilter is developed as part of a publish–subscribe system that allows users to submit a set of queries that are to be executed against streaming XML pages. By compiling the queries into an automaton to index all provided patterns, the system is able to efficiently find the list of all users who submitted a query that matches the current document. In our integrated version of YFilter, the detection patterns (either for Web application type or level) will be submitted as queries; when a document satisfy a query, the system will stop processing the document against all remaining queries (in contrast to the standard behavior of YFilter), as we do not need more than one match. In addition, to deal with predicates that are present in our language but that YFilter does not support, we modify the system so that additional filters can be tested before validating a match.

```
Input: a URL u, sets of detection patterns D and crawling actions A
if alreadyCrawled(u) then
    if hasChanged(u) then
        markedActions ← detectAndMarkStructuralChanges(u, A);
        newActions ← alignCrawlingActions(u, D, markedActions);
        addToKnowledgeBase(newActions);
```

Algorithm 1. Adaptation to template change (recrawl of a Web application)

6 Adaptation to Template Change

We describe here how the AAH adapts to changes in the structure of Web applications. Structural changes w.r.t. the knowledge base may come from varying versions of the content management system, or from alternative templates proposed by the CMS or developed for specific Web applications. The AAH determines when a change has occurred and tries adapting patterns and actions.

We deal with two different cases of adaptation: first, when (part of) a Web application has been crawled before the template change and a recrawl is carried out after that (a common situation in real-world crawl campaigns); second, when crawling a new Web application that matches the Web application type detection patterns but for which (some of) the actions are inapplicable.

Recrawl of a Web Application. We first consider the case when part of a Web application has been crawled successfully using the patterns and actions of the knowledge base. The template of this Web application then changes (because of an update of the content management system, or a redesign of the site) and it is recrawled. Our core adaptation technique relearns appropriate crawling actions for each crawlable object; the knowledge base is then updated by adding newly relearned actions to it.

As later described in Sect. 7, crawled Web pages with their Web objects and metadata are stored in the form of RDF triples into a RDF store. Our proposed system detects structural changes for already crawled Web applications by looking for the content (stored in the RDF store) in the Web pages with the crawling actions used during the previous crawl. If the system fails to extract the same content with these actions, the structure of the Web site has changed.

Algorithm 1 gives a high-level view of the template adaptation mechanism in the case of a recrawl. It first checks whether a given URL has already been crawled by calling the *alreadyCrawled* Boolean function, which just looks for the existence of the URL in the RDF store. An already crawled Web page will then be checked for structured changes with the *hasChanged* Boolean function.

Structural changes are detected by searching for already crawled content (URLs corresponding to navigation actions, Web objects, etc.) in a Web page by using the existing and already learned crawling actions (if any) for the corresponding Web application level. The *hasChanged* function takes care of the fact that failure to extract deleted information should not be considered as a

```
Input: a URL u and a sets of crawling actions A
if not alreadyCrawled(u) then
    for a ∈ A do
        if hasExtractionFailed(u, a) then
            relaxedExpressions ← getRelaxedExpressions(a);
            for candidate ∈ relaxedExpressions do
                if not hasExtractionFailed(u, candidate) then
                    addToKnowledgeBase(candidate);
                    break;
```

Algorithm 2. Adaptation to template change (new Web application)

structural change. For instance, a Web object such as a Web forum's *comment* that was crawled before may not exist anymore.

In the presence of structural changes, the system calls the *detectAndMarkStructuralChanges* function which detects inapplicable crawling actions and mark them as "failed". All crawling actions which are marked as failed will be aligned according to structural changes. The *alignCrawlingActions* function will relearn the failed crawling actions.

Crawl of a New Web Application. We are now in the case where we crawl a completely new Web applications whose template is (slightly) different from that present in the knowledge base. We assume that the Web application type detection patterns fired, but either the application level detection patterns or the crawling actions do not work on this specific Web application.

Let us first consider the case where the Web application level detection pattern works. Recall that there are two classes of Web application levels: intermediate and terminal. We make the assumption that on intermediate levels, crawling actions (that are solely navigation actions) do not fail – on that level, navigations actions are usually fairly simple (they typically are simple extensions of the application level detection patterns, e.g., `//div[contains(@class,'post')]` for the detection pattern and `//div[contains(@class,'post')]//a/@href` for the navigation action). In our experiments we never needed to adapt them. We leave the case where they might fail to future work. On the other hand, we consider that both navigation actions and extraction actions from terminal pages may need to be adapted. The main steps of the adaptation algorithm are described in Algorithm 2. *getRelaxedExpressions* creates two set of relaxed expression (for best-case and worst-case). For each set, different variations of crawling action will be generated by relaxing predicates and tag names, enumerated by the number of relaxation needed (simple relaxations come first). Tag names are replaced with existing tag names of the DOM tree so that the relaxed expression matches. When relaxing an attribute name inside a predicate, the AAH only suggests candidates that would make the predicate true; to do that, the AAH first collects all possible attributes and their values from the page. We favor relaxations that use parts from crawling actions in the knowledge base for other Web

application types of the same general category (e.g., Web forum). The system orders expressions by the number of required relaxations (best-case ones first). Any expression which succeeds in the extraction will still be tested with a few more pages of the same Web application level before being added to the knowledge base for future crawling.

If the system does not detect the Web application level, then the crawling strategy cannot be initiated. First, the system tries adapting the detection pattern before fixing crawling actions. The idea is here the same as in the previous part: the system collect all candidate attributes, values, tag names from the knowledge base for the detected Web application type (e.g., WordPress) and then creates all possible combinations of relaxed expressions, ordered by the amount of relaxation, and test them one by one until one that works is found. To illustrate, assume that the candidate set of attributes and values are: `@class='post'`, `@id='forum'`, `@class='blog'` with candidate set of names `article` ,`div`, etc. The set of relaxed expression will be generated by trying out each possible combination: `// article [contains(@class,'post')]`, `// article [contains(@id,'forum')]`, `// article [contains(@class,'blog')]`, etc.

7 System

The application-aware helper is implemented in Java. On startup, the system first loads the knowledge base and indexes detection patterns using a YFilter [24] implementation adapted from the one available at `http://yfilter.cs.umass.edu/`. Once the system receives a crawling request, it first makes a lookup to the YFilter index to detect the Web application type and level. If the Web application type is not detected, the AAH applies the adaptation strategy to find a relaxed match as previously described. If no match is found (i.e., if the Web application is unknown), a generic extraction of links is performed.

When the Web application is successfully detected, the AAH loads the corresponding crawling strategy from the knowledge base and crawls the Web application accordingly, possibly using the adaptation strategy. Crawled Web pages are stored in the form of WARC [25] files – the standard preservation format for Web archiving – whereas structured content (individual Web objects with their semantic metadata) is stored in an RDF store. The knowledge base is potentially updated with new detection patterns or crawling actions.

The AAH is integrated with Heritrix [6], the open-source crawler[2] developed by the Internet Archive. In the crawl processing chain, the AAH replaces the conventional link extraction module. Crawling actions determined by the AAH are fed back into the URL queue of Heritrix.

The open-source AAH code and the list of all sites in our experimental dataset are available at `http://perso.telecom-paristech.fr/~faheem/aah.html`.

[2] `http://crawler.archive.org/`

8 Experiments

We present in this section experimental performance of our proposed system on its own and with respect to a baseline crawler, GNU wget[3] (since the scope of the crawl is quite simple – complete crawling of specific domain names – wget is as good as Heritrix here).

Experiment Setup. To evaluate the performance of our system, we have crawled 100 Web applications (totaling nearly 3.3 millions Web pages) of two forms of social Web sites (Web forum and blog), for three specific content management system (vBulletin, phpBB, and WordPress). The Web applications of type WordPress (33 Web applications, 1.1 million of Web pages), vBulletin (33 Web applications, 1.2 million of Web pages) and phpBB (34 Web applications, 1 million Web pages) were randomly selected from three different sources:

1. `http://rankings.big-boards.com/`, a database of popular Web forums.
2. A dataset related to European financial crisis.
3. A dataset related to the *Rock am Ring* music festival in Germany.

The second and third datasets were collected in the framework of the ARCOMEM project[4]. In these real-world datasets corresponding to specific archival tasks, 68% of the seed URLs of Web forum type belongs to either vBulletin or phpBB, which explains while we target these two CMSs. WordPress is also a prevalent CMS: the Web as a whole has over 61 million Wordpress sites [26] out of a number of blogs indexed by Technorati [27] of around 133 million. Moreover, Wordpress has a 48% market share of the top 100 blogs [28]. All 100 Web applications were both crawled using wget and the AAH. Both crawlers are configured to retrieve only HTML documents, disregarding scripts, stylesheets, media files, etc.

The knowledge base is populated with detection patterns and crawling actions for one specific version of the three considered CMSs (other versions will be handled by the adaptation module). Adding a new Web application type to the knowledge base takes a crawl engineer of the order of 30 minutes.

Performance Metrics. The performance of the AAH will be mainly measured by evaluating the number of HTTP requests made by both systems vs the amount of *useful* content retrieved. Evaluating the number of HTTP requests is easy to perform by simply counting requests made by both crawlers. Coverage of useful content is more subjective and we use the following proxies:

1. Counting the amount of textual content that has been retrieved. For that, we compare the proportion of 2-grams (sequences of two consecutive words) in the crawl result of both systems, for every Web application.
2. Counting the number of external links (i.e., hyperlinks to another domain) found in the two crawls. The idea is that external links are a particularly important part of the content of a Web site.

[3] `http://www.gnu.org/software/wget/`
[4] `http://www.arcomem.eu/`

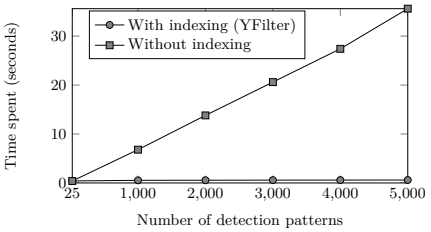

Fig. 2. Performance of the detection module

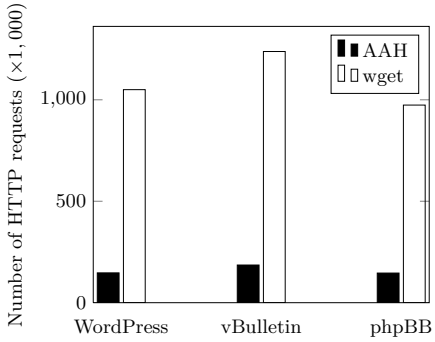

Fig. 3. Total number of HTTP requests used to crawl the dataset

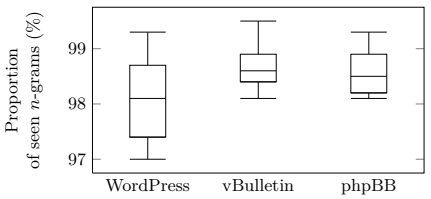

Fig. 4. Box chart of the proportion of seen n-grams for the three considered CMSs. We show in each case the minimum and maximum values (whiskers), first and third quartiles (box) and median (horizontal rule).

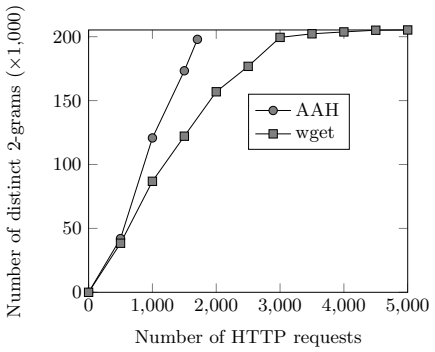

Fig. 5. Crawling `http://www.rockamring-blog.de/`

Efficiency of Detection Patterns. We first briefly discuss the use of YFilter to speed up the indexing of detection patterns. In Fig. 2 we show the time required to determine Web application type in a synthetically generated knowledge base as the number of Web application types grows up to 5,000, with or without using YFilter indexing. The system takes a time linear in the number of detection patterns when indexing is turned off, taking up to several dozens of seconds. On the other hand, detection time is essentially constant with YFilter activated.

Crawl Efficiency. We compare the number of HTTP requests required by both crawlers to crawl each set of Web applications of the same type in Fig. 3. Notice how the application-aware helper makes much fewer requests (on average 7 times fewer) than a regular blind crawl. Indeed, for blog-like Web sites, a regular crawler make redundant HTTP requests for the same Web content, accessing to a post by tag, author, year, chronological order, etc. In a Web forum, many requests end up being search boxes, edit areas, print view of a post, areas protected by authentication, etc.

Table 1. Coverage of external links in the dataset crawled by the AAH

CMS	External links	External links (w/o boilerplate)
WordPress	92.7%	99.8%
vBulletin	90.5%	99.5%
phpBB	92.1%	99.6%

Crawl Effectiveness. The crawling results, in terms of coverage of useful content, are summarized in Fig. 4 and in Table 1. Figure 4 presents the distribution of the proportion of n-grams crawled by the AAH with respect to those of the full crawl. Not only are the numbers are generally very high (for the three types, the median is greater than 98%), but the results are also very stable, with a very low variance: the worst coverage score on our whole dataset is greater than 97% (typically, lower scores are achieved for small Web sites where the amount of boilerplate text such as menus or terms of use remains non negligible). This hints at the statistical significance of the results.

The proportion of external links covered by the AAH is given in Table 1. The application-aware helper has ignored nearly 10 percent of external links since every page may use widgets, such as those of Facebook, Amazon, etc., with URLs varying from one page to another. Once we have excluded boilerplate with defined set of patters, we see that more than 99.5% of the external links are present in the content crawled by the AAH.

To reach a better understanding of how an application-aware crawl enfolds, we plot in Fig. 5 the number of distinct 2-grams discovered by the AAH and wget during one crawl (in this particular case, of a given WordPress blog), as the number of requests increase. We see that the AAH directly targets the interesting part of the Web application, with a number of newly discovered 2-grams that grows linearly with the number of requests made, to reach a final level of 98% 2-gram coverage after 1,705 requests. On the other hand, wget discovers new content with a lower rate, and, especially, spends the last 2/5 of its requests discovering very few new 2-grams.

Comparison to iRobot. The iRobot system [11] that we discussed in Sect. 2 is not available for testing because of intellectual property reasons. The experiments of [11] are somewhat limited in scope, since only 50,000 Web pages are considered, over 10 different forum Web sites (to compare with our evaluation, on 3.3 million Web pages, over 100 different forum or blog Web sites). To compare the AAH to iRobot, we have crawled one of the same Web forum used in [11]: `http://forums.asp.net/` (over 50,000 Web pages). The completeness of content of the AAH (in terms of both 2-grams and external links, boilerplate excluded) is over 99 percent; iRobot has a coverage of *valuable pages* (as evaluated by a human being) of 93 percent on the same Web application. The number of HTTP requests for iRobot is claimed in [11] to be 1.73 times less than a

Table 2. Examples of structural pattern changes: desktop vs mobile version of `http://www.androidpolice.com/`

Desktop version	Mobile version
div[@class='post_title']/**h3**/a	div[@class='post_title']/**h2**/a
div[@class='post_info']	div[@class='post_author']
div[@class='post_content']	div[@class='content']

regular Web crawler; on the `http://forums.asp.net/` Web application, the AAH makes 10 times fewer requests than wget does.

Adaptation When Recrawling a Web Application. To test our adaptation technique in the case of a recrawl of a Web application in a realistic environment (without having to wait for Web sites actually to change), we have considered sites that have both a desktop and mobile version with different HTML content. These sites use two different templates to present what is essentially the same content. We simulated a recrawl by first crawling the Web site with a `User-Agent:` HTTP header indicating a regular Web spider (the desktop version is then served) and then recrawling the mobile version using a mobile browser `User-Agent:`.

Our system was not only able to detect the structural changes from one version to another, but also, using already crawled content, to fix the failed crawling actions. Table 2 presents one exemplary Web application that has both a desktop and mobile versions, with a partial list of the structural changes in the patterns across the two versions. Our system was able to automatically correct these structure changes in both navigation and extraction, reaching a perfect agreement between the content extracted by the two crawls.

Adaptation for a New Web Application. As stated earlier, we have experimented our system with 100 Web applications, starting from a straightforward knowledge base containing information about one specific version of the three considered content management systems. Among the 100 applications, 77 did not require any adaptation, which illustrates that many Web applications share common templates. The 23 remaining ones had a structure that did not match the crawling actions in the knowledge base; the AAH has applied adaptation successfully to these 23 cases. Most of the adaptation consisted in relaxing the class or id attribute rather than replacing the tag name of an element. When there was a tag name change, it was most often from span to div to article or vice versa, which is fairly straightforward to adapt. There was no case in the dataset when more than one relaxation for a given step of an XPath expression was needed; in other words, only best-case relaxed expressions were used. In 2 cases, the AAH was unable to adapt all extraction actions, but navigation actions still worked or could be adapted, which means the Web site could still be crawled, but some structured content was missing.

9 Conclusions

In Web archiving, scarce resources are bandwidth, crawling time, and storage space rather than computation time [5]. We have shown how application-aware crawling can help reduce bandwidth, time, and storage (by requiring less HTTP requests to crawl an entire Web application, avoiding duplicates) using limited computational resources in the process (to apply crawling actions on Web pages). Application-aware crawling also helps adding semantics to Web archives, increasing their value to users.

Our work can be extended in several ways, that we shall explore in future work. First, we can enrich the pattern language we use to allow for more complex detection and extraction rules, moving to a full support of XPath or even more powerful Web navigation languages allowing to crawl complex Web applications making use of AJAX or Web forms. There is a trade-off, however, between the expressive power of the language and the simplicity of template adaptations. Second, we want to move towards an automatically constructed knowledge base of Web applications, either by asking a human being to automatically annotate the part of a Web application to extract or crawl, using semi-supervised machine learning techniques, or even by discovering in an unsupervised manner new Web application types by comparing the structure of different Web sites, determining the optimal way to crawl them by sampling, in the spirit of iRobot [11].

Acknowledgment. This work was funded by the European Union's Seventh Framework Program (FP7/2007–2013) under grant agreement 270239 (ARCOMEM).

References

1. Jupp, E.: Obama's victory tweet 'four more years' makes history. The Independent (November 2012), `http://ind.pn/RF5Q6O`
2. Coleman, S.: Blogs and the new politics of listening. The Political Quarterly 76(2) (2008)
3. Mulvenon, J.C., Chase, M.: You've Got Dissent! Chinese Dissident Use of the Internet and Beijing's Counter Strategies. Rand Publishing (2002)
4. Giles, J.: Internet encyclopaedias go head to head. Nature 438 (2005)
5. Masanès, J.: Web archiving. Springer (2006)
6. Sigurðsson, K.: Incremental crawling with Heritrix. In: IWAW (2005)
7. Faheem, M.: Intelligent crawling of Web applications for Web archiving. In: WWW PhD Symposium (2012)
8. Chakrabarti, S., van den Berg, M., Dom, B.: Focused crawling: A new approach to topic-specific Web resource discovery. Comp. Networks 31(11-16) (1999)
9. Gibson, D., Punera, K., Tomkins, A.: The volume and evolution of Web page templates. In: WWW (2005)
10. Guo, Y., Li, K., Zhang, K., Zhang, G.: Board forum crawling: A Web crawling method for Web forums. In: Web Intelligence (2006)
11. Cai, R., Yang, J.M., Lai, W., Wang, Y., Zhang, L.: iRobot: An intelligent crawler for Web forums. In: WWW (2008)

12. Ying, H.M., Thing, V.: An enhanced intelligent forum crawler. In: CISDA (2012)
13. Edmonds, J.: Optimum branchings. J. Res. Nat. Bureau Standards 71B (1967)
14. Kolari, P., Finin, T., Joshi, A.: SVMs for the blogosphere: Blog identification and splog detection. In: AAAI (2006)
15. Kushmerick, N.: Regression testing for wrapper maintenance. In: AAAI (1999)
16. Chidlovskii, B.: Automatic repairing of Web wrappers. In: WIDM (2001)
17. Meng, X., Hu, D., Li, C.: Schema-guided wrapper maintenance for Web-data extraction. In: WIDM (2003)
18. Lerman, K., Minton, S.N., Knoblock, C.A.: Wrapper maintenance: A machine learning approach. J. A. I. Res. (2003)
19. Lim, S.J., Ng, Y.K.: An automated change-detection algorithm for HTML documents based on semantic hierarchies. In: ICDE (2001)
20. Artail, H., Fawaz, K.: A fast HTML Web page change detection approach based on hashing and reducing the number of similarity computations. Data Knowl. Eng. (2008)
21. Ferrara, E., Baumgartner, R.: Automatic wrapper adaptation by tree edit distance matching. In: Hatzilygeroudis, I., Prentzas, J. (eds.) Combinations of Intelligent Methods and Applications. SIST, vol. 8, pp. 41–54. Springer, Heidelberg (2011)
22. Gulhane, P., Madaan, A., Mehta, R., Ramamirtham, J., Rastogi, R., Satpal, S., Sengamedu, S.H., Tengli, A., Tiwari, C.: Web-scale information extraction with vertex. In: ICDE (2011)
23. W3C: Web application description language (2009), `http://www.w3.org/Submission/wadl/`
24. Diao, Y., Altinel, M., Franklin, M.J., Zhang, H., Fischer, P.: Path sharing and predicate evaluation for high-performance XML filtering. ACM TODS (2003)
25. ISO: ISO 28500:2009, Information and documentation – WARC file format
26. WordPress: WordPress sites in the world (2012), `http://en.wordpress.com/stats/`
27. The Future Buzz: Social media, Web 2.0 and internet stats (2009), `http://goo.gl/H0FNF`
28. Royal Pingdom: WordPress completely dominates top 100 blogs (2012), `http://goo.gl/eifRJ`