Crawl intelligent et adaptatif d'applications web pour l'archivage du web

Muhammad Faheem, Pierre Senellart

Institut Mines–Télécom; Télécom ParisTech; CNRS LTCI
46 rue Barrault
75634 Paris Cedex 13, France
firstname.lastname@telecom.paristech.fr

RÉSUMÉ. *Les sites web sont par nature dynamiques, leur contenu et leur structure changeant au fil du temps; de nombreuses pages sur le web sont produites par des systèmes de gestion de contenu (CMS). Les outils actuellement utilisés par les archivistes du web pour préserver le contenu du web collectent et stockent de manière aveugle les pages web, en ne tenant pas compte du CMS sur lequel le site est construit ni du contenu structuré de ces pages web. Nous présentons dans cet article un* application-aware helper *(AAH) qui s'intègre à une chaine d'archivage classique pour accomplir une collecte intelligente et adaptative des applications web. Parce que l'AAH est conscient des applications web actuellement collectées, il est capable de raffiner la liste des URL à traiter et d'ajouter à l'archive de l'information sémantique sur le contenu extrait. Afin de traiter les changements possibles de structure des applications web, notre AAH inclut un module d'adaptation qui rend la collecte résistante aux petits changements de structure du site web. Nous démontrons la valeur de notre approche en comparant la sortie et l'efficacité du AAH par rapport à des robots web traditionnels, également en présence de changements de structure.*

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). 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 and whatever structured content is contained in Web pages. 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. 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 structure changes in Web applications, our AAH includes an adaptation module that makes crawling resilient to small changes in the structure of Web sites. 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.*

MOTS-CLÉS : *système de gestion de contenu, crawling, application web, archivage du web, XPath.*

KEYWORDS: *content management system, crawling, Web application, Web archiving, XPath.*

DOI:10.3166/ISI.19.4.61-86

1. Introduction

1.1. Archivage du web social

Le World Wide Web est devenu un système de publication actif et une source riche en informations, grâce aux contributions de centaines de millions d'individus, qui utilisent le web social comme médium pour diffuser leurs émotions, pour publier du contenu, pour discuter de sujets politiques, pour partager des vidéos, pour poster des commentaires et aussi pour donner leur opinion personnelle dans des discussions en cours. Une partie de cette expression publique s'accomplit sur les sites de réseaux sociaux et de partage social (Twitter, Facebook, Youtube, etc.), une partie sur des sites indépendants reposant sur des systèmes de gestion de contenu (*content management systems* ou CMS, incluant les blogs, les wikis, les sites d'actualités avec systèmes de commentaires, les forums web). Le contenu publié sur cette gamme d'*applications web* n'inclut pas seulement les divagations d'utilisateurs lambda du web, mais également des informations qui sont d'ores-et-déjà remarquables ou qui seront précieuses aux historiens de demain. Barack Obama a ainsi d'abord annoncé sa réélection de 2012 comme président des États-Unis sur Twitter (Jupp, 2012) ; les blogs sont de plus en plus utilisés par les hommes politiques pour diffuser leur programme et pour être à l'écoute des citoyens (Coleman, 2008) ; les forums web sont devenus un mode commun, pour les dissidents politiques, de discussion de leurs opinions (Mulvenon, Chase, 2002) ; des initiatives comme le projet Polymath [1] transforment les blogs en des tableaux interactifs collaboratifs pour la recherche (Nielsen, 2011) ; les wikis construits par les utilisateurs, comme Wikipedia, contiennent des informations dont la qualité est au niveau des ouvrages de référence traditionnels (Giles, 2005).

Parce que le contenu du web est distribué, change perpétuellement et est souvent stocké dans des plates-formes propriétaires sans garantie d'accès à long terme, il est critique de préserver ces contenus précieux pour les historiens, journalistes, ou sociologues des générations futures. Ceci est l'objectif du domaine de l'*archivage du web* (Masanès, 2006), qui traite de la découverte, du crawl, du stockage, de l'indexation, et de la mise à disposition à long terme des données du web.

1.2. Archivage dépendant de l'application

Les crawlers d'archivage actuels, tels que Heritrix d'Internet Archive (Sigurðsson, 2005), fonctionnent de manière conceptuellement simple. Ils partent d'une liste *graine* d'URL à stocker dans une file (p. ex., l'URL de départ peut être la page principale d'un site). Les pages web sont ensuite récupérées de cette pile l'une après l'autre (en respectant l'éthique du crawl, en limitant le nombre de requêtes par serveur), stockées telles qu'elles, et des hyperliens en sont extraits. Si ces liens sont dans la portée de la tâche d'archivage, les URL nouvellement extraites sont ajoutées à la file. Ce processus

1. http://polymathprojects.org/

s'arrête après une durée déterminée ou quand aucune nouvelle URL intéressante ne peut être trouvée.

Cette approche ne répond pas aux défis du crawl d'applications web modernes : la nature de l'application web crawlée n'est pas prise en compte pour décider de la stratégie de crawl ou du contenu stocké ; les applications web avec du contenu dynamiques (p. ex., les web forums, les blogs) peuvent être crawlées de manière inefficace, en termes de nombre de requêtes HTTP requises pour archiver un site donné ; le contenu stocké dans l'archive peut être redondant, et n'a typiquement aucune structure (il consiste en des fichiers HTML plats), ce qui rend l'accès à l'archive pénible.

Le but de ce travail est d'adresser ces défis en introduisant une nouvelle approche *dépendante de l'application* pour le crawl web d'archivage. Notre système, l'*application-aware helper* (ou AAH en bref) repose sur une base de connaissances d'applications web connues. Une *application web* est n'importe quelle application basée sur HTTP qui utilise le web et les technologies des navigateurs web pour publier de l'information en utilisant un modèle spécifique. Nous nous focalisons en particulier sur les aspects sociaux du web, qui dépendent largement des contenus engendrés par les utilisateurs, de l'interaction sociale, et de la mise en réseau, ainsi qu'on le trouve sur les forums web, les blogs, ou sur les sites de réseaux sociaux. Notre AAH proposé ne récupère que la partie importante du contenu d'une application web (c'est-à-dire, le contenu qui sera précieux dans une archive web) et évite les doublons, les URL inintéressantes et les parties de la page qui ont des buts de présentation. De plus l'AAH extrait des pages du web des contenus individuels (tels que le contenu d'un message de blog, son auteur, son estampille temporelle) qui peuvent être stockés en utilisant des technologies du web sémantique pour un accès plus riche et plus sémantique à l'archive.

Pour illustrer, considérons l'exemple d'un forum web, s'appuyant sur un système de gestion de contenu comme vBulletin [2]. Côté serveur, les fils de discussion et les messages du forum sont stockés dans une base de données. Fréquemment, l'accès à deux URL différentes va résulter en le même contenu ou en du contenu qui se recouvre partiellement. Par exemple, on peut accéder à un message d'une utilisatrice donnée à la fois à travers la vue classique des messages d'un forum sous la forme d'un fil de discussion, ou par la liste de toutes ses contributions telles qu'affichées sur son profil utilisateur. Cette redondance signifie qu'une archive construite par un crawler web classique contiendra de l'information dupliquée, et que de nombreuses requêtes au serveur ne produiront aucune information nouvelle. Dans des cas extrêmes, le crawler peut tomber dans un *piège à robots* car il a un nombre virtuellement infini de liens à crawler. Il y a également plusieurs liens inutiles comme ceux vers une version imprimable d'une page, ou vers une publicité, liens qu'il vaudrait mieux éviter durant la création d'une archive. Au contraire, le client web qui est conscient de l'information à crawler peut déterminer un chemin optimal pour crawler l'ensemble des messages d'un forum, en évitant toute requête inutile, et peut stocker les messages individuels,

2. http://www.vbulletin.com/

avec leurs auteurs et estampille, dans une forme structurée dont les archivistes et autres utilisateurs de l'archive pourront bénéficier.

1.3. Changement de structure

Les applications web sont par nature dynamiques ; non seulement leur contenu change avec le temps, mais leur structure et apparence changent également. Les systèmes de gestion de contenu fournissent différents thèmes qui peuvent être utilisés pour engendrer des articles de wiki, des messages de blog ou de forum, etc. Ces systèmes fournissent habituellement une manière de changer ce modèle (qui conduit ultimement à un changement de structure des pages web) sans altérer le contenu en information, pour s'adapter aux besoins d'un site spécifique. Le rendu peut également changer quand une nouvelle version du CMS est installée.

Tous ces changements de rendu aboutissent à des changements possibles dans l'arbre DOM de la page web, souvent mineurs. Cela rend plus difficile de reconnaître et de traiter de manière intelligente toutes les instances d'un système de gestion de contenu donné, car il est sans espoir de décrire manuellement toutes les variations possibles d'un thème dans une base de connaissances des applications web. Un autre but de notre travail est de produire une approche de collecte intelligente qui est résistante aux changements mineurs de modèles et, en particulier, qui adapte automatiquement son comportement à ces changements, mettant à jour sa connaissance des CMS à cette occasion. Notre technique d'adaptation s'appuie à la fois sur une relaxation des motifs de crawl et d'extraction présents dans la base de connaissances et sur la comparaison des versions successives de la même page web.

1.4. Plan

Nous présentons à la suite de cette introduction un état de l'art du crawl d'applications web et de l'adaptation au changement de structure. Après avoir donné quelques définitions préliminaires en section 3, nous décrivons notre base de connaissances d'applications web en section 4. La méthodologie que notre application-aware helper implémente est ensuite présentée en section 5. Nous discutons du problème spécifique de l'adaptation au changement de structure et les algorithmes liés en section 6, avant de couvrir les questions d'implémentation et d'expliquer comment l'AAH s'utilise conjointement à un crawler en section 7. Nous comparons enfin l'efficacité (en termes de temps et de résultats) de notre AAH par rapport à l'approche classique de crawl pour crawler des blogs et forums web en section 8.

Cet article est une version étendue (et traduite) d'un travail publié à la conférence ICWE 2013 (Faheem, Senellart, 2013). Des idées initiales ayant conduit à ce travail ont également été d'abord présentées comme un article d'un séminaire de doctorants dans (Faheem, 2012).

2. Travaux connexes

Nous présentons dans cette section les travaux connexes : tout d'abord sur le crawl du web dans son ensemble, puis plus spécifiquement sur le crawl de forum, pour lequel notre AAH est particulièrement adapté, comme nous le détaillerons. Nous mentionnons également les travaux liés à la détection d'applications web, à l'adaptation d'extracteurs, et à l'extraction de données structurées depuis des sites web.

2.1. Crawl du web

Le crawl du web est un problème bien étudié qui comporte encore des défis. Un état de l'art du domaine de l'archivage du web et du crawl pour l'archivage est disponible dans (Masanès, 2006).

Un crawler *dirigé* (*focused*) parcourt le web en suivant un ensemble prédéfini de thèmes (Chakrabarti *et al*., 1999). Cette approche est une façon différente de la nôtre d'influencer le comportement d'un crawler, qui n'est pas basée sur la structure des applications web comme nous le visons, mais sur le contenu des pages web. Notre approche n'a pas le même but que le crawl dirigé : elle a plutôt vocation à mieux archiver des sites web basés sur des CMS connus. Les deux stratégies d'amélioration d'un crawler conventionnel sont donc complémentaires.

Le contenu des applications web ou des *systèmes de gestion de contenu* est organisé en fonction d'un modèle de rendu (les composants du modèle incluent par exemple les barres latérales d'une page web, la barre de navigation, un en-tête ou pied de page, la zone de contenu principal, etc.). Parmi les nombreux travaux sur l'extraction de ce modèle, (Gibson *et al*., 2005) ont réalisé une analyse de l'étendue du contenu engendré par de tels modèles sur le web. Ils ont découvert que 40 à 50 % du contenu du web (en 2005) est basé sur des modèles, c'est-à-dire, fait partie d'une application web. Leurs résultats suggéraient également que cette proportion augmente à un taux de 6 à 8 % par an. Ces travaux sont une forte indication du bénéfice à gérer le crawl d'applications web de manière spécifique.

2.2. Crawl de forum

Même si le crawl basé sur l'application web en toute généralité n'a pas encore été traité, il y a eu des efforts portant sur l'extraction de contenu dans des forums web (Guo *et al*., 2006 ; Cai *et al*., 2008).

La première approche, appelée *Board Forum Crawling (BFC)* exploite la structure organisée des forums web et simule le comportement de l'utilisateur dans le processus d'extraction. BFC résout le problème efficacement, mais est confronté à des limitations car il repose sur des règles simples et peut uniquement traiter des forums qui ont une structure spécifique.

La deuxième approche (Cai *et al*., 2008), cependant, ne dépend pas de la structure spécifique d'un web forum donné. Le système iRobot assiste le processus d'extraction en extrayant la carte de l'application web crawlée. Cette carte est construite en crawlant de manière aléatoire quelques pages de l'application. Ce processus aide à identifier les régions répétitives riches qui sont par la suite regroupées suivant leurs modèles (Zheng *et al*., 2007). Après la création de la carte, iRobot obtient la structure du forum web sous la forme d'un graphe orienté consistant en des nœuds (les pages web) et des arêtes orientées (les liens entre pages). Par la suite, une analyse de chemin est réalisée pour fournir un chemin de traversée optimale conduisant au processus d'extraction, cela afin d'éviter les pages doublons et invalides. Un travail ultérieur (Ying, Thing, 2012) a analysé iRobot et a identifié un certain nombre de limitations. Ying et Thing (2012) étendent le système original par un ensemble de fonctionnalités : une meilleure découverte de l'arbre couvrant minimal (Edmonds, 1967), une meilleure mesure du coût d'une arête dans le processus de crawl comme estimation de sa profondeur approchée dans le site et un raffinement de la détection des doublons.

iRobot (Cai *et al*., 2008 ; Ying, Thing, 2012) est probablement le travail le plus proche du nôtre. Cependant, l'AAH que nous proposons s'applique à n'importe quelle application web, à partir du moment où elle est décrite dans notre base de connaissances. À la différence aussi par rapport à (Cai *et al*., 2008 ; Ying, Thing, 2012), où l'analyse de la structure d'un forum doit être faite indépendamment pour chaque site, l'AAH exploite le fait que plusieurs sites partagent le même système de gestion de contenu. Notre système extrait également de l'information structurée et sémantique des pages web, tandis que iRobot produit des fichiers HTML bruts et laisse l'extraction à des travaux futurs. Nous donnons finalement en section 8 une comparaison des performances de iRobot par rapport à AAH pour mettre en valeur les meilleures performances de notre approche. D'un autre côté, iRobot a pour but d'être une manière pleinement automatique de crawler un forum web, tandis que l'AAH s'appuie sur une base de connaissances (construite manuellement mais maintenue automatiquement) d'applications web ou de CMS.

2.3. Détection d'applications web

Comme nous l'expliquerons, notre approche repose sur un mécanisme générique de détection du type d'applications web couramment crawlé. Pour ce sujet également, de la littérature existe dans le cas particulier des blogs ou forums. En particulier, (Kolari *et al*., 2006) utilise des *support vector machines (SVM)* pour détecter si une page donnée est une page de blog. Les SVM (Boser *et al*., 1992 ; Osuna *et al*., 1997) sont largement utilisées pour les tâches de classification de texte. Dans (Kolari *et al*., 2006), les SVM sont entraînées en utilisant des vecteurs de caractéristiques formés du multi-ensemble des mots du contenu ou des n-grammes, et de quelques autres caractéristiques spécifiques à la détection de blog, telles que le multi-ensemble des URL liées et des ancres. L'entropie relative est utilisée pour la sélection de caractéristiques.

Plus généralement, quelques travaux (Amitay *et al*., 2003 ; Lindemann, Littig, 2007 ; 2006) visent à identifier une catégorie générale (p. ex., blog, académique, personnel) pour un site web donné, en utilisant des systèmes de catégorisation basés sur des caractéristiques structurelles des pages web qui visent à détecter les *fonctionnalités* de ces sites web. Ce n'est pas directement applicable à notre cadre, tout d'abord parce que la catégorisation est très grossière, et ensuite parce que ces techniques fonctionnent au niveau du site web (p. ex., en s'appuyant sur la page principale) et non au niveau des pages individuelles.

2.4. Adaptation d'extracteur

Le problème de l'adaptation d'un extracteur d'informations web à des changements (mineurs) dans la structure de pages ou sites web a reçu une certaine attention dans la communauté académique. Un premier travail est celui de (Kushmerick, 1999) qui a proposé une approche d'analyse des pages web et de l'information déjà extraite, afin de détecter des changements de structure. Une méthode de « vérification d'extracteur » est introduite, cette méthode vérifiant si un extracteur arrête d'extraire des données ; si oui, un superviseur humain est informé afin d'entraîner à nouveau l'extracteur. (Chidlovskii, 2001) a introduit des règles grammaticales et logiques pour automatiser la maintenance d'extracteurs, en supposant qu'il n'y a que de petits changements dans la structure des pages web. (Meng *et al*., 2003) ont suggéré une maintenance d'extracteurs guidée par le schéma, appelée SG-WRAM, qui repose sur l'observation que même quand la structure des pages change, certaines caractéristiques sont conservées, comme les motifs syntaxiques, les hyperliens, les annotations, et l'information extraite elle-même. (Lerman *et al*., 2003) ont développé un système d'apprentissage pour réparer des extracteurs vis-à-vis de petits changements de balisage. Le système qu'ils ont proposé vérifie d'abord l'extraction des pages web, et si celle-ci échoue, relance l'apprentissage d'extracteur. (Raposo *et al*., 2007) collectent des résultats de requêtes valides durant l'utilisation de l'extracteur ; quand des changements structurels ont lieu dans des sources web, ils utilisent ces résultats comme entrées pour générer un nouvel ensemble d'entraînement étiqueté qui peut être à nouveau utilisé pour l'apprentissage d'extracteur.

Notre technique d'adaptation de modèle est inspirée par les travaux précédemment cités : nous vérifions si notre extracteur échoue, et si c'est le cas, nous essayons de le corriger en faisant l'hypothèse de changements mineurs sur le site, et parfois en utilisant le contenu déjà crawlé. Une différence principale avec les travaux existants est que notre approche est également applicable à des sites web complètement nouveaux, qui n'ont jamais été crawlés, mais qui partagent juste le même système de gestion de contenu et un modèle similaire.

2.5. Extraction de données depuis des blogs, forums, etc.

Un certain nombre de travaux (Xia *et al*., 2011 ; Ferrara, Baumgartner, 2010 ; Lim, Ng, 2001 ; Artail, Fawaz, 2008) ont pour but l'extraction automatique, non supervisée,

depuis des pages web engendrées par un CMS, en recherchant des structures répétées et, habituellement, en utilisant des techniques d'alignement ou d'appariement d'arbres. Ceci est hors de la portée de notre approche, où nous supposons que nous disposons d'une base pré-existante d'applications web. Gulhane *et al*. (2011) ont introduit le système d'apprentissage d'extracteur Vertex. Vertex extrait de l'information structurée depuis des pages web basées sur un modèle. Le système classifie les pages en utilisant des vecteurs de n-grammes, apprend des règles d'extraction, détecte des changements du site et réapprend les règles qui ne fonctionnent plus. Vertex détecte les changements du site en surveillant quelques pages d'exemple du site. N'importe quel changement structurel peut résulter en des changements dans les vecteurs de n-grammes et peut rendre les règles apprises inapplicables. Dans notre système, nous ne surveillons pas des pages web sélectionnées mais adaptons dynamiquement le comportement aux pages au fur et à mesure que nous les crawlons. Notre crawler s'adapte également à différentes versions d'un système de gestion de contenu, trouvé sur différents sites web, plutôt que simplement à un type de page web spécifique.

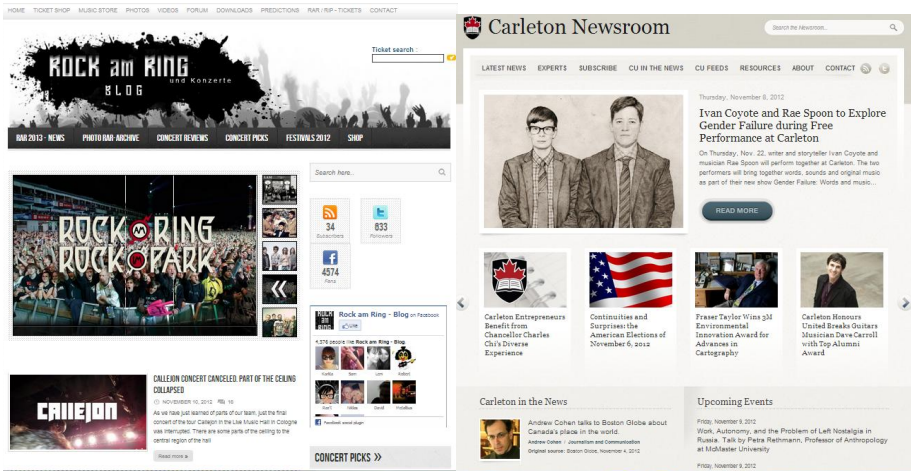
3. Préliminaires

Cette section introduit la terminologie que nous utiliserons tout au long de l'article.

Une *application web* est n'importe quelle application ou site qui utilise les standards du web tels que HTML et HTTP pour publier de l'information sur le web via un modèle donné, d'une manière accessible aux navigateurs web. Des exemples sont les forums web, les sites de réseaux sociaux, les services de géolocalisation, etc.

Un *type d'application web* est le système de gestion de contenu ou la pile de technologies côté serveur (p. ex., vBulletin, WordPress, le CMS propriétaire de Flickr) qui exécute cette application web et fournit une interaction avec elle. Plusieurs applications web différentes peuvent partager le même type d'application web (tous les forums vBulletin utilisent vBulletin), mais certains types d'application web peuvent être spécifiques à une application web donnée (p. ex., le CMS de Twitter est spécifique à ce site). Il peut y avoir des différences significatives dans l'apparence des applications web du même type ; comparer par exemple les figures 1a et 1b, qui présentent deux sites utilisant WordPress.

Le contenu présenté par les applications web est habituellement stocké dans les bases de données et des modèles prédéfinis sont utilisés pour engendrer des pages web riches en données. Pour le crawl intelligent, notre AAH doit savoir non seulement distinguer les types d'applications web, mais aussi les différents types de pages web qui peuvent être produites par un type d'application web donné. Par exemple, un logiciel de forum web peut engendrer des pages de contenu de diverses sortes, comme *liste de forums*, *liste de fils*, *liste de messages*, *message individuel*, *profil utilisateur*. Nous appelons une telle sorte de modèle spécifique le *niveau* de l'application web.



(a) Rock am Ring blog

(b) Carleton Newsroom

(c) Niveau intermédiaire

(d) Niveau terminal

Figure 1. Exemples de pages web d'applications web

Nous utilisons un sous-ensemble simple du langage XPath (W3C, 1999) pour décrire des *motifs* dans le DOM des pages web qui servent soit à identifier le type ou niveau d'application web, soit à déterminer les *actions* de navigation ou d'extraction à appliquer à cette page web. Une grammaire pour le sous-ensemble que nous considérons est donnée en figure 2. Essentiellement, nous autorisons uniquement les axes de navigation vers le bas et des prédicats très simples qui comparent des chaînes de caractères. La sémantique de ces expressions est la sémantique standard (W3C, 1999). Sauf si précisé différemment, un « // » est implicite : une expression peut s'appliquer à n'importe quel nœud quelle que soit sa profondeur dans l'arbre DOM. Dans ce qui suit, une *expression XPath* désigne toujours une expression de ce sous-langage.

Un *motif de détection* est une règle pour détecter les types et niveaux d'applications web, basée sur le contenu des pages web, des méta-données HTTP, des composants de l'URL. Il est implémenté comme une expression XPath sur un document virtuel qui contient la page web HTML ainsi que les autres méta-données HTTP.

⟨expr⟩ ::= ⟨étape⟩ | ⟨étape⟩ "/" ⟨expr⟩
⟨étape⟩ "//" ⟨expr⟩
⟨étape⟩ ::= ⟨testNoeud⟩ | ⟨étape⟩ "[" ⟨prédicat⟩ "]"
⟨testNoeud⟩ ::= *élément* | "@" *élément* | "*" | "@*" | "text()"
⟨prédicat⟩ ::= "contains(" ⟨valeur⟩ "," *chaîne* ")" |
⟨valeur⟩ "=" *chaîne* | *entier* | "last()"
⟨valeur⟩ ::= *élément* | "@" *élément*

Figure 2. Syntaxe BNF du fragment XPath utilisé. Les lexèmes suivants sont utilisés : élément *est un identifiant XML valide (ou NMTOKEN (W3C, 2008)) ;* chaîne *est une chaîne de caractères XPath entourée par des guillemets simples ou doubles (W3C, 1999) ;* entier *est un entier positif*

Une *action de crawl* est une expression XPath sur un document HTML qui indique quelle action accomplir sur une page web donnée. Les actions de crawl peuvent être de deux sortes : les *actions de navigation* pointent vers des URL à ajouter dans la file de crawl ; les *actions d'extraction* pointent vers des objets sémantiques individuels à extraire de la page web (p. ex., estampille temporelle, message de blog, commentaire). Un exemple d'action de navigation pour suivre certains types de liens est : div[contains(@class,'post')]//h2[@class='post-message']//a/@href.

L'AAH distingue deux degrés principaux de niveaux d'applications web : les pages *intermédiaires*, comme les listes de forums, les listes de fils de discussion, ne peuvent être associées qu'à des actions de navigation ; les *pages terminales*, comme la liste des messages d'un fil de discussion, peuvent être associées à la fois à des actions de navigation et d'extraction. L'idée est que le crawleur navigue parmi les pages intermédiaires jusqu'à trouver une page terminale, et seul le contenu de cette page terminale est extrait ; on peut également naviguer à partir de la page terminale, p. ex., quand il y a un découpage d'un fil de discussion par pages numérotées. Pour illustrer, les figures 1c et 1d montrent, respectivement, des pages intermédiaires et terminales dans une application de forum web.

Étant donnée une expression XPath e, une *expression affaiblie* pour e est une expression où une ou plusieurs des transformations suivantes ont été accomplies :

— un prédicat a été supprimé ;
— un lexème *élément* a été remplacé par un autre lexème *élément*.

Une *expression affaiblie meilleur-cas* pour e est une expression où au plus l'une des transformations a été effectuée pour chaque étape de e. Une *expression affaiblie pire-cas* pour e est une expression où un nombre arbitraire de transformations ont été effectuées pour chaque étape de e.

Pour illustrer, posons
e =div[contains(@class,'post')]//h2[@class='post-message'].
Des exemples d'expressions affaiblies meilleurs-cas sont

div[contains(@class,'post')]//h2
ou bien
div[contains(@class,'post')]//h2[@id='post-content'];
en revanche, on remarque que
div[contains(@class,'post')]//div[@id='post-content']
est un exemple d'expression affaiblie pire-cas.

4. Base de connaissances

L'AAH est assisté d'une *base de connaissances* des types d'application web qui décrit comment crawler un site web intelligemment. Ces connaissances spécifient comment détecter des applications web spécifiques et quelles actions de crawl doivent être exécutées. Les types sont organisés de manière hiérarchique, des catégorisations générales aux instances spécifiques (sites web) de cette application web. Par exemple, les sites web de médias sociaux peuvent être classés en blogs, forums web, microblogs, réseaux de partage de vidéos, etc. Ensuite, nous pouvons décomposer encore ces types d'applications web sur la base des systèmes de gestion de contenu sur lequel ils sont basés. Ainsi, WordPress, Movable Type, sont des exemples de système de gestion de contenu de blog, tandis que phpBB et vBulletin sont des exemples de systèmes de gestion de contenu de forum web. Cette base de connaissances décrit également les différents niveaux sous un type d'application web ; en fonction de ceux-ci, nous pouvons définir différentes actions de crawl qui doivent être exécutées pour chaque niveau.

La base de connaissances est spécifiée dans un langage déclaratif, afin d'être aisément partagée et mise à jour, si possible maintenue par des non-programmeurs et aussi peut-être automatiquement apprise à partir d'exemples. Le W3C a standardisé un langage de description d'application web (*Web Application Description Language* ou *WADL*) (W3C, 2009) qui permet de décrire les ressources des applications basées sur HTTP dans un format interprétable par un programme. WADL est utilisé pour décrire l'ensemble des ressources, leurs relations les unes avec les autres, les méthodes qui peuvent être appliquées sur chaque ressource, les formats de représentation de ressources. WADL peut être un candidat de composant et de format d'export pour notre base de connaissances, mais ne remplit pas tous nos besoins : en particulier, il n'y a aucune place pour la description des motifs de reconnaissance d'applications web. Par conséquent, notre base de connaissances est décrite dans un format XML approprié, de manière bien adaptée à la structure arborescente de la hiérarchie d'applications web et de niveaux de pages.

4.1. Type et niveau d'application web

Pour chaque type d'application web et pour chaque niveau, la base de connaissances contient un ensemble de motifs de détection qui permet de reconnaître si une page donnée est de ce type ou de ce niveau.

Prenons l'exemple du système de gestion de contenu de forum web vBulletin, qui peut ainsi être identifié en cherchant une référence à un script JavaScript du nom de `vbulletin_global.js` avec le motif de détection :
script[contains(@src,'vbulletin_global.js')]
Les pages vBulletin du type « liste de forums » sont identifiées [3] quand elles correspondent au motif a[@class="forum"]/@href.

4.2. Actions de crawl

De manière similaire, pour chaque type et niveau d'application web, un ensemble d'actions de navigation et d'extraction (dans ce dernier cas, seulement dans le cas des niveaux terminaux) est fourni. Les actions de navigation pointent vers des liens (typiquement dans un attribut a/@href) à suivre ; les extractions d'action, également associés à des types sémantiques, pointent vers du contenu à extraire.

5. Application-aware helper (AAH)

Notre position principale dans cet article est que des techniques de crawl différentes doivent être appliquées à différents types d'applications web. Cela veut dire des stratégies de crawl différentes pour différentes formes de sites web sociaux (blogs, wikis, réseaux sociaux, marque-pages sociaux, microblogs, partage de musique, forums web, partage de photos, partage de vidéos, etc.), pour des systèmes de gestion de contenu spécifiques (p. ex., WordPress, phpBB) et pour des sites spécifiques (p. ex., Twitter, Facebook). Les figures 1a et 1b sont des exemples d'applications web qui sont basées sur le CMS WordPress. Notre approche sera de détecter le type d'application web (type général, système de gestion de contenu ou site) en cours de traitement par le crawler et le type de pages web à l'intérieur de cette application web (p. ex., un profil utilisateur sur un réseau social) et de décider d'actions de crawl (suivre un lien, extraire du contenu structuré) en conséquence. Le crawler proposé est suffisamment intelligent pour crawler et stocker tous les commentaires liés à un message de blog en un endroit, même si ces commentaires sont distribués sur plusieurs pages web.

L'AAH détecte l'application web et le type de page web avant de décider quelle stratégie de crawl est appropriée pour l'application web donnée. Plus précisément, l'AAH fonctionne dans l'ordre suivant :

1. il détecte le type d'application web ;
2. il détecte le niveau d'application web ;
3. il exécute les actions de crawl pertinentes : extraire le résultat des actions d'extraction et ajouter le résultat des actions de navigation à la file d'URL.

3. L'exemple est simplifié pour les besoins de la présentation ; en réalité il faut gérer les différents rendus que vBulletin peut produire.

Détection du type et niveau d'application web

L'AAH charge les motifs de détection de type d'application web depuis la base de connaissances et les exécute sur l'application web. Si le type est détecté, le système exécute tous les motifs de détection de niveau de l'application web jusqu'à trouver un résultat. Le nombre de motifs de détection pour détecter un type d'application web va croître avec l'ajout de connaissances sur de nouvelles applications web. Pour optimiser cette détection, le système a besoin de maintenir un index de ces motifs.

Dans ce but, nous avons intégré le système YFilter (Diao *et al*., 2003), un système de filtrage pour expressions XPath basé sur un automate non-déterministe. Nous lui avons appliqué des petites modifications, notamment pour l'indexation efficace des motifs de détection, afin de trouver rapidement les types et niveaux d'application web pertinents. YFilter est développé comme partie d'un système de publication–souscription qui permet aux utilisateurs de soumettre un ensemble de requêtes qui doivent être exécutées sur des pages XML en flux. En compilant les requêtes dans un automate pour indexer tous les motifs fournis, le système est capable de trouver efficacement la liste de tous les utilisateurs ayant soumis une requête qui est valide sur le document courant. Dans notre version intégrée de YFilter, les motifs de détection (que ce soit pour le type ou le niveau d'application web) sont soumis comme des requêtes ; quand un document satisfait à une requête, le système arrête de tester les autres requêtes sur ce document (contrairement au comportement standard de YFilter), car nous n'avons pas besoin de plus d'une correspondance.

6. Adaptation au changement de modèle

Deux types de changements peuvent se produire dans une page web : le contenu peut changer et la structure peut changer. Des changements dans la note d'un film, un nouveau commentaire sous un message de blog, la suppression d'un commentaire, etc., sont des exemples de changement de contenu web. Ces types de changements sont faciles à identifier en comparant simplement la page web courante avec une version récemment crawlée. Un exemple de changement de modèle ou de structure est un changement dans le modèle de présentation d'un site web, qui est plus complexe à identifier et à adapter. Notre approche de l'adaptation apporte une solution à ce défi.

Les changements de structure par rapport à ce qui est dans la base de connaissances peut venir de versions différentes d'un système de gestion de contenu, ou de thèmes alternatifs proposés par ce CMS ou développés pour des applications web spécifiques. Si le modèle d'une page web n'est pas celui donné dans la base de connaissances des applications web, les motifs de détection des applications web et les actions de crawl peuvent échouer. L'AAH a pour but de déterminer quand un changement a eu lieu et d'adapter les motifs et actions en conséquence. Au cours de ce processus, il maintient également automatiquement la base de connaissances avec les nouveaux motifs et actions découverts.

Les motifs de détection pour déterminer le type d'une application web sont la plupart du temps détectés en recherchant une référence à un fichier de script, une feuille de style, des méta-données HTTP (tels que des noms de cookies) ou même un contenu textuel (p. ex., un fragment textuel « Powered by »), qui sont en général robustes aux changements de structure. Nous allons faire l'hypothèse dans la suite que les motifs de détection du type d'application web n'échouent jamais. Nous n'avons constaté aucun cas où cela se produisait dans nos expériences. En revanche, nous considérons que les motifs de détection du niveau de l'application web et les actions de crawl peuvent devenir inapplicables après un changement de modèle.

Nous traitons deux cas différents d'adaptation : tout d'abord, quand (une partie d')une application web a déjà été crawlée avant le changement et qu'un nouveau crawl est fait après le changement (une situation courante dans les campagnes de crawl réelles) ; deuxièmement, quand une nouvelle application web est crawlée, qu'elle correspond aux motifs de détection des types d'application web, mais pour laquelle certaines actions sont inapplicables.

6.1. Crawl d'une application web déjà crawlée

Nous considérons d'abord le cas où une partie d'une application web a été crawlée avec succès en utilisant les motifs et les actions de la base de connaissances. Le modèle de cette application web change ensuite (en raison d'une mise à jour du système de gestion de contenu, ou d'un changement de conception du site) et l'application est crawlée à nouveau. La base de notre technique d'adaptation est d'apprendre à nouveau les actions de crawl appropriées pour chaque objet qui peut être crawlé ; la base de connaissances est ensuite mise à jour en y ajoutant les actions nouvellement apprises (comme les actions existantes peuvent toujours fonctionner sur une autre instance de ce type d'application web, nous les gardons dans la base).

Comme nous le décrivons plus tard en section 7, les pages web crawlées sont stockées, avec leurs objets web et méta-données, sous la forme de triplets RDF dans une base RDF. Notre système détecte les changements de structure pour les applications web déjà crawlées en recherchant le contenu (stocké dans la base RDF) dans les pages web avec les actions utilisées durant le crawl précédent. Si le système ne parvient pas à extraire le contenu avec les mêmes actions, cela veut dire que la structure du site web a changé.

L'algorithme 1 donne une vue de haut niveau du mécanisme d'adaptation à un changement de modèle dans ce cas. Le système traite chaque page web pertinente d'une application web. Toute page web qui a déjà été crawlée mais ne peut maintenant être crawlée sera envoyée à l'algorithme 1 pour adaptation de structure. Cet algorithme ne répare que les actions de crawl qui ont échoué pour des pages déjà crawlées ; cependant, comme les actions de crawl nouvellement obtenues sont ajoutées à la base de connaissances, le système sera capable de traiter des pages de la même application web qui n'ont pas encore été crawlées. Il faut remarquer que dans tous les cas l'algorithme n'essaie pas d'apprendre à nouveau les actions de crawl ayant échoué pour chaque

Entrées : URL u, ensembles de motifs de détection d'applications web D et d'actions de crawl A
si *déjàCrawlé*(u) **alors**
  si *aChangé*(u) **alors**
    actionsMarquées $\leftarrow$ *détecteChangementsStructurels*(u, A);
    nouvellesActions $\leftarrow$ *aligneActionsCrawl*$(u, D,$ *actionsMarquées*$)$;
    ajouteBaseConnaissance(*nouvellesActions*);

Algorithme 1. Adaptation au changement de modèle (crawl d'une application web déjà crawlée)

page web, mais commence plutôt par vérifier si les actions de crawl déjà réparées pour le même niveau d'application web fonctionnent toujours.

L'algorithme 1 commence par vérifier si une URL donnée a déjà été crawlée en appelant la fonction booléenne *déjàCrawlé* qui vérifie simplement si l'URL existe dans la base RDF. Une page web déjà crawlée sera ensuite analysée pour détecter des changements de structure avec la fonction booléenne *aChangé*.

Les changements de structure sont détectés en cherchant du contenu déjà crawlé (URL correspondant à des actions de navigation, objets web, etc.) dans une page web en utilisant les actions de crawl existantes et déjà apprises (le cas échéant) pour le niveau d'application web. La fonction *aChangé* prend en compte le fait qu'échouer à extraire de l'information supprimée ne doit pas être considéré comme un changement de modèle. Par exemple, un objet web tel qu'un commentaire dans un forum web qui a été crawlé auparavant peut ne plus exister. Ce scénario peut se produire car un administrateur, voire l'auteur du commentaire lui même, a pu le supprimer du forum. Pour cette raison, le système vérifiera toujours l'existence d'un objet web, avec toutes ses méta-données, dans la version courante d'une page web ; si l'objet web a disparu, il n'est pas pris en compte pour la détection de changements de modèle.

En présence de changements de structure, le système appelle la fonction *détecte-ChangementsStructurels* qui détecte les actions inapplicables et les marque comme « échouées ». La fonction *détecteChangementsStructurels* retourne un ensemble d'actions de crawl marquées. Toutes les actions de crawl qui sont marquées comme échouées vont être adaptées aux changements de structure. La fonction *aligneActionsCrawl* apprend une nouvelle version des actions de crawl ayant échoué.

Si un motif de détection de niveau d'application web échoue également, alors le système applique une approche similaire à celle décrite en section 6.2.2 pour adapter le motif de détection de niveau.

6.2. Crawl d'une nouvelle application web

Nous sommes maintenant dans le cas où nous crawlons une application web complètement nouvelle dont le modèle a (légèrement) changé par rapport à celui contenu dans la base de connaissances. Nous supposons que les motifs de détection du type

d'application ont fonctionné, mais que soit les motifs de détection de niveau, soit les actions de crawl ne fonctionnent pas sur cette application web précise. Dans cette situation, nous ne pouvons pas nous appuyer sur du contenu précédemment crawlé.

Considérons d'abord le cas où le motif de détection de niveau d'application web fonctionne.

6.2.1. Le niveau d'application web est détecté

Rappelons qu'il y a deux types de niveaux d'application web : intermédiaire et terminal. Nous supposons qu'aux niveaux intermédiaires, les actions de crawl (qui sont uniquement des actions de navigation) n'échouent pas – à ce niveau, les actions de navigation sont habituellement assez simples (elles sont typiquement de simples extensions des motifs de détection de niveau, p. ex., `//div[contains(@class,'post')]` pour le motif de détection et `//div[contains(@class,'post')]//a/@href` pour l'action de navigation). Dans nos expériences nous n'avons jamais eu besoin de les adapter. Nous laissons le cas où elles peuvent échouer à des travaux ultérieurs. En revanche, nous considérons qu'à la fois les actions de navigation et les actions d'extraction des pages terminales peuvent nécessiter une adaptation.

Entrées : URL u et ensemble d'actions de crawl A
si *non déjàCrawlé*(u) **alors**
  pour $a \in A$ **faire**
    si *extractionEchouée*(u, a) **alors**
      expressionsAffaiblies $\leftarrow$ *affaiblit*(a);
      pour *candidat* $\in$ *expressionsAffaiblies* **faire**
        si *non extractionEchouée*$(u,$ *candidat*$)$ **alors**
          ajouteBaseConnaissance(*candidat*);
          quitteBoucle ;

Algorithme 2. Adaptation au changement de modèle (nouvelle application web)

Les étapes principales de l'algorithme d'adaptation sont décrites dans l'algorithme 2. Le système vérifie d'abord l'applicabilité des actions de crawl existantes et corrige ensuite celles ayant échoué. La fonction *affaiblit* crée deux ensembles d'expressions affaiblies (meilleur-cas et pire-cas). Pour chaque ensemble, les différentes variations des actions de crawl vont être engendrées en affaiblissant les prédicats et noms d'éléments, et énumérées en fonction du nombre d'affaiblissements requis (les affaiblissements simples venant d'abord). Les noms d'éléments sont remplacés avec les noms d'élément existants dans l'arbre DOM afin que l'expression DOM ait une correspondance. Quand un nom d'attribut est affaibli à l'intérieur d'un prédicat, l'AAH engendre uniquement les candidats qui sont susceptibles de rendre ce prédicat vrai ; à cette fin, l'AAH collecte d'abord tous les attributs possibles et leurs valeurs depuis la page.

Quand l'action de crawl a pour préfixe un motif de détection, nous ne touchons pas à ce préfixe mais affaiblissons uniquement la deuxième partie. Par exemple, si une expression comme `div[contains(@class,'post')]//h2[@class='post-title']` échoue à extraire le titre d'un message et que `div[contains(@class,'post')]`

est le motif de détection qui a été déclenché, nous essayons des affaiblissements de la deuxième partie de l'expression, p. ex., remplacer @class avec @id, 'post-title' avec 'post-head', h2 avec div, etc. Nous favorisons les relaxations qui utilisent des parties d'actions de crawl de la base de connaissances pour d'autres types d'applications web de la même catégorie générale (p. ex., forum web).

Une fois que le système a engendré tous les affaiblissements possibles, il essaie d'abord les motifs meilleur-cas et, s'ils ne fonctionnent pas, les motifs pire-cas. Plus généralement, les expressions sont ordonnées par le nombre d'affaiblissements requis. Toute expression qui réussit dans l'extraction sera testée sur quelques autres pages du même niveau d'application web avant d'être ajoutée dans la base de connaissances pour un crawl futur.

6.2.2. Le niveau d'application web n'a pas été détecté

Si le système ne détecte pas le niveau d'application web, la stratégie de crawl ne peut être lancée. Le système essaie d'abord d'adapter les motifs de détection avant de corriger les actions de crawl. L'idée est ici la même que dans la partie précédente : le système collecte tous les attributs, valeurs, noms d'éléments candidats depuis la base de connaissances du type d'application web détecté (p. ex., WordPress) et crée ensuite toutes les combinaisons possibles d'expressions affaiblies, ordonnées par le nombre d'affaiblissements, et les teste une par une jusqu'à en trouver une qui fonctionne.

Pour illustrer, supposons que l'ensemble d'attributs et valeurs candidats sont : @class='post', @id='forum', @class='blog' avec les ensembles candidats de noms d'éléments article , div, etc. L'ensemble des expressions affaiblies va être engendré en essayant chaque combinaison possible : // article [contains(@class,'post')]
// article [contains(@id,'forum')]
// article [contains(@class,'blog')]
et similairement pour les autres noms d'éléments.

Après la collection de l'ensemble des expressions relâchées, le système va essayer de détecter le niveau d'application web en testant chaque expression relâchée et, si le système détecte finalement le niveau, il applique la technique d'adaptation décrite précédemment. Si le système ne détecte pas le niveau d'application web, l'adaptation échoue.

7. Système

L'application-aware helper est implantée en Java. Au démarrage, le système charge la base de connaissances et indexe les motifs de détection en utilisant une implantation de YFilter (Diao *et al.*, 2003) adaptée de celle disponible à http://yfilter.cs.umass.edu/. Une fois que le système reçoit une requête de crawl, il consulte d'abord l'index YFilter pour détecter le type et niveau d'application web. Si le type d'application web n'est pas détecté, l'AAH applique la stratégie d'adaptation pour trouver une correspondance

affaiblie comme précédemment décrit. Si aucune correspondance n'est trouvée (c'est-à-dire, si l'application web est inconnue), une extraction générique des liens est réalisée.

Quand l'application web est détectée avec succès, l'AAH charge la stratégie de crawl correspondante de la base de connaissances et crawle l'application web en fonction de celle-ci, éventuellement en utilisant la stratégie d'adaptation. Les pages web crawlées sont stockées sous la forme de fichiers WARC (ISO, 2009) – le format standard de préservation pour l'archivage web – tandis que le contenu structuré (les objets web individuels avec leurs méta-données sémantiques) est stocké dans une base RDF. Pour le passage à l'échelle et la disponibilité de l'archive, les fichiers WARC sont mis sur une grappe Hadoop [4] sous HDFS, et la base RDF que nous utilisons, H2RDF (Papailiou *et al.*, 2012), est implantée au-dessus de HBase (The Apache Software Foundation, 2012). La base de connaissances est potentiellement mise à jour avec de nouveaux motifs de détection ou des actions de crawl.

L'AAH est intégré au sein d'Heritrix (Sigurðsson, 2005), le crawler libre [5] developpé par Internet Archive. Dans la chaîne de traitement du crawl, l'AAH remplace le module conventionnel d'extraction de liens. Les actions de crawl déterminées par l'AAH sont renvoyées dans la file des URL à crawler de Heritrix.

Le code de l'AAH (ainsi que la liste de tous les sites de notre jeu de données expérimental) est disponible sur http://perso.telecom-paristech.fr/~faheem/aah.html sous licence GPL 3.0.

8. Expériences

Nous présentons dans cette partie la performance expérimentale du système que nous proposons, en elle-même et par comparaison à un crawler traditionnel, GNU wget [6]. Comme la portée du crawl est ici très simple – un crawl complet d'un ensemble de noms de domaines – wget est ici aussi bien que Heritrix.

8.1. Protocole expérimental

Pour évaluer la performance de notre système, nous avons crawlé 100 applications web (avec un total de près de 3,3 millions de pages web) de deux formes de sites web sociaux (forum web et blog), pour trois systèmes de gestion de contenu (vBulletin, phpBB et WordPress). Les applications web de type WordPress (33 applications web, 1,1 millions de pages web), vBulletin (33 applications web, 1,2 millions de pages web) et phpBB (34 applications web, 1 million de pages web) ont été sélectionnées aléatoirement de trois sources différentes :

1. http://rankings.big-boards.com/, une base de données de forums web populaires.

4. http://hadoop.apache.org/
5. http://crawler.archive.org/
6. http://www.gnu.org/software/wget/

2. Un jeu de données sur le sujet de la crise financière en europe.
3. Un jeu de données lié au festival de musique *Rock am Ring* en Allemagne.

Les deuxième et troisième jeux de données ont été collectés dans le cadre du projet ARCOMEM (ARCOMEM Project, 2011–2013). Dans ces jeux de données du monde réel correspondant à des tâches d'archivage spécifiques, 68 % des URL initiales de type forum sont des sites que font tourner soit vBulletin soit phpBB, ce qui explique que nous nous intéressons à ces deux CMS. WordPress est également un CMS prévalent : le web dans son ensemble a plus de 61 millions de sites Wordpress (WordPress, 2012) sur un total de blogs indexés par Technorati (The Future Buzz, 2009) d'à peu près 133 millions. De plus, Wordpress a une part de marché de 48% des 100 blogs les plus visités (Royal Pingdom, 2012). Chacune des 100 applications web a été crawlée en utilisant wget et l'AAH. Les deux crawlers sont configurés pour récupérer uniquement les documents HTML, sans tenir compte des scripts, des feuilles de style, des fichiers média, etc.

La base de connaissances est peuplée de motifs de détection et d'actions de crawl pour une version spécifique des trois systèmes de gestion de contenu considérés (les autres versions seront traitées par le module d'adaptation). Ajouter un nouveau type d'application à la base de connaissances (sous la forme de l'édition d'un document XML) prend à un ingénieur de crawl de l'ordre de 30 minutes.

8.2. Métriques de performance

La performance de l'AAH sera principalement mesurée en évaluant le nombre de requêtes HTTP réalisées par les deux systèmes par rapport au volume de contenu *utile* récupéré. Notons que comme wget récupère aveuglément l'intégralité du site web, toutes les URL crawlées par l'AAH seront aussi crawlées par wget.

Évaluer le nombre de requêtes HTTP est facile, il suffit de compter les requêtes réalisées par les deux crawlers. La couverture du contenu utile est une notion plus subjective ; comme il est impossible de demander une évaluation utilisateur étant donnés les volumes que nous considérons, nous utilisons les alternatives suivantes :

1. Compter le volume de contenu textuel récupéré. À cette fin, nous comparons la proportion des bigrammes (séquences de deux mots consécutifs) dans le résultat du crawl des deux systèmes, pour chaque application web.
2. Compter le nombre de liens externes (c'est-à-dire, d'hyperliens vers un autre domaine) trouvés dans les deux crawls. L'idée est que les liens externes sont une partie importante du contenu du site web.

8.3. Passage à l'échelle de la détection du type

Nous discutons d'abord brièvement de l'utilisation de YFilter pour améliorer l'indexation des motifs de détection. Dans la figure 3 nous montrons le temps requis pour déterminer le type d'application web dans une base de connaissances engendrée de

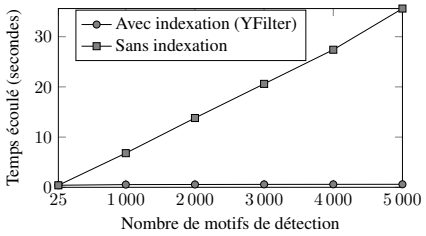


Figure 3. Performance du module de détection

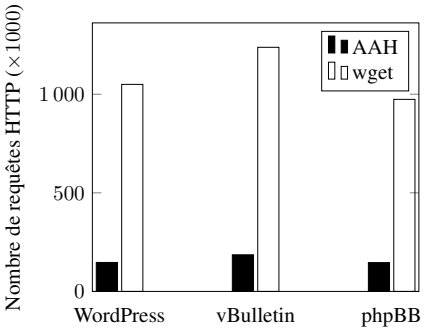


Figure 4. Nombre total de requêtes HTTP utilisées pour crawler le jeu de données

manière synthétique quand le nombre de types d'application croît jusque 5 000, avec ou sans l'indexation par YFilter. Le système prend un temps linéaire dans le nombre de motifs de détection quand l'indexation est désactivée, prenant jusqu'à plusieurs dizaines de seconde. En revanche, le temps de détection est quasi constant quand YFilter est activé.

8.4. Efficacité du crawl en nombre de requêtes

Nous comparons le nombre de requêtes HTTP requises par les deux crawlers pour crawler chaque ensemble d'applications web de même type en figure 4. Notons que l'AAH fait beaucoup moins de requêtes (en moyenne 7 fois moins) qu'un crawl aveugle classique. En effet, pour les sites web de type blog, un crawler classique réalise un nombre abondant de requêtes HTTP redondantes pour le même contenu web, accédant à un message par tag, auteur, année, ordre chronologique, etc. Dans un forum web, de nombreuses requêtes se trouvent pointer vers des fonctionnalités de recherche, des zones d'édition, une vue pour impression d'un message, des zones protégées par authentification, etc.

8.5. Qualité du crawl

Tableau 1. Couverture des liens externes dans le jeu de données crawlé par l'AAH

CMS	**Liens externes**	**Liens externes (sans modèle)**
WordPress	92,7 %	99,8 %
vBulletin	90,5 %	99,5 %
phpBB	92,1 %	99,6 %

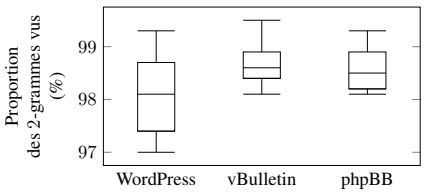


Figure 5. Boîte à moustaches des bigrammes vus pour les trois CMS considérés. Nous montrons dans chaque cas les valeurs minimum et maximum (moustaches), le premier et troisième quartiles (boîte) et la médiane (ligne horizontale).

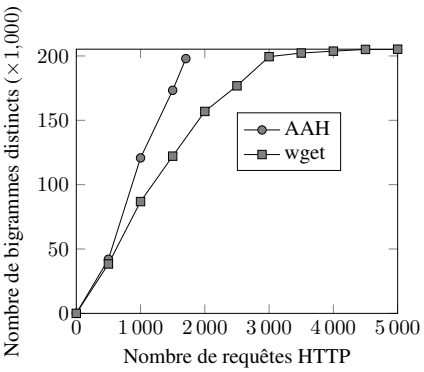


Figure 6. Crawl de http://www.rockamring-blog.de/

Les résultats de crawl, en termes de couverture du contenu utile, sont résumés dans la figure 5 et dans le tableau 1. La figure 5 présente la distribution de la proportion de bigrammes crawlés par l'AAH en fonction de ceux du crawl complet. Non seulement les chiffres sont en général très haut (pour les trois types, la médiane est supérieure à 98 %) mais les résultats sont également très stables, avec une très faible variance ; le pire score de couverture sur notre jeu de données complet est supérieur à 97 % (habituellement, des scores plus faibles viennent de petits sites web où la proportion de texte faisant partie du modèle comme des noms de menu ou les conditions d'utilisation du site restent non négligeables). Cela indique que les résultats sont significatifs.

La proportion de liens externes couverte par l'AAH est donnée dans le tableau 1. L'AAH a ignoré près de 10 % des liens externes car chaque page peut utiliser des composants, comme ceux de Facebook, d'Amazon, etc., avec des URL variant d'une page à l'autre ; une fois ces composants exclus avec un ensemble de motifs fixé, nous voyons que plus de 99,5 % des liens externes sont présents dans le contenu crawlé par l'AAH.

Pour mieux comprendre comment un crawl de l'AAH se déroule, nous montrons en figure 6 le nombre de bigrammes distincts découverts par l'AAH et wget durant un crawl (dans ce cas particulier, un blog WordPress), alors que le nombre de requêtes augmente. Nous voyons que l'AAH cible directement la partie intéressante d'une application web, avec un nombre de bigrammes nouvellement découverts qui croît linéairement avec le nombre de requêtes effectuées, pour atteindre un niveau final de 98 % de couverture de bigrammes après 1 705 requêtes. D'autre part, wget découvre du contenu nouveau à un rythme plus lent et, en particulier, passe les dernières 2/5 de ses requêtes à découvrir très peu de nouveau bigrammes.

8.6. Comparaison à iRobot

Le système iRobot (Cai *et al*., 2008) dont nous avons parlé en section 2 n'est pas disponible à des fins de test pour des raisons de propriété intellectuelle. Les expériences de (Cai *et al*., 2008) sont assez limitées en portée, avec seulement $50\,000$ pages web considérées, sur 10 sites de forum web différents (à comparer à notre évaluation, sur $3,3$ millions de pages web, et 100 sites de forum et blog différents). Pour comparer AAH et iRobot, nous avons crawlé l'un des mêmes forums web utilisé dans (Cai *et al*., 2008) : http://forums.asp.net/ (plus de 50,000 pages web). La couverture du contenu par l'AAH (en termes de bigrammes et de liens externes, modèle exclu) est de plus de 99 % ; iRobot a une couverture des *pages de valeur* (estimées par un humain) de 93 % sur la même application web. Le nombre de requêtes HTTP pour iRobot est dite être dans (Cai *et al*., 2008) 1,73 fois moins qu'un crawler web classique ; sur l'application web http://forums.asp.net/, l'AAH fait 10 fois moins de requêtes que wget.

8.7. Adaptation pour un crawl d'une application déjà crawlée

Pour tester notre technique d'adaptation dans le cas d'un crawl d'une application web déjà crawlée dans un cadre réaliste (sans avoir à attendre que les sites web changent effectivement), nous avons considéré des sites ayant à la fois une version « ordinateur de bureau » et une version mobile, avec du contenu HTML différent. Ces sites utilisent deux modèles différents pour présenter ce qui est essentiellement le même contenu. Nous avons simulé le double crawl en crawlant d'abord le premier site avec un en-tête HTTP `User-Agent:` indiquant un robot web classique (la version ordinateur de bureau est alors servie) puis en crawlant la version mobile avec un `User-Agent:` de navigateur mobile.

Tableau 2. Exemples de changements de motifs structurels : version ordinateur de bureau et mobile de http://www.androidpolice.com/

Version ordinateur de bureau	Version mobile
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']

Notre système a non seulement pu détecter les changements de structure d'une version à l'autre mais aussi, en utilisant le contenu déjà crawlé, a pu corriger les actions de crawl échouant. Le tableau 2 présente une application web exemple qui a à la fois une version ordinateur de bureau et mobile, avec une liste partielle des changements structurels dans les motifs des deux versions. Notre système a été capable de corriger automatiquement ces changements de structure en termes à la fois de navigation et d'extraction, atteignant une parfaite concordance entre les contenus extraits dans les deux crawls.

8.8. Adaptation à une nouvelle application web

Ainsi qu'indiqué précédemment, nous avons expérimenté notre système sur 100 applications web, en partant d'une base de connaissances contenant des informations sur une version spécifique des trois systèmes de gestion de contenu considéré. Parmi les 100 applications, 77 n'ont pas requis d'adaptation, ce qui illustre le fait que de nombreuses applications web partagent les mêmes modèles. Les 23 applications restantes avaient une structure qui différait de celle indiquée par les actions de crawl de la base de connaissances ; l'AAH a appliqué la technique d'adaptation avec succès sur ces 23 cas. La plupart des adaptations consistaient en affaiblissant l'attribut @class ou @id plutôt qu'en remplaçant le nom d'un élément. Quand il y avait un changement de nom d'élément, c'était la plupart du temps de span vers div vers article et vice-versa, ce qui est assez simple à adapter. Il n'y a eu aucun cas dans le jeu de données où plus d'un affaiblissement pour chaque étape d'une expression XPath fût requis ; en d'autres termes, seulement des expressions meilleur-cas furent utilisées. Dans deux cas, l'AAH ne fut pas capable d'adapter l'ensemble des actions d'extraction, mais les actions de navigation fonctionnaient toujours ou ont pu être adaptées, ce qui veut dire que le site web a pu être crawlé, mais qu'une partie du contenu structuré était manquant de l'extraction.

Comme exemple d'affaiblissement, dans l'application http://talesfromanopenbook.wordpress.com/, l'extraction du titre de message par l'action div[@class='post']/h2[@class='post-title'] échoua mais l'affaiblissement div[@class='post']/h2[@class='storytitle'] fut trouvé.

9. Conclusions

Dans l'archivage du web, les ressources rares sont la bande passante, le temps de crawl et l'espace de stockage, plutôt que le temps de calcul (Masanès, 2006). Nous avons montré que le crawl d'applications web basé sur les applications peut aider à réduire la bande passante, le temps et l'espace de stockage (en nécessitant moins de requêtes HTTP pour crawler une application web entière, évitant les doublons) tout en utilisant des ressources de calcul limitées (pour appliquer les actions de crawl sur les pages web). Le crawl basé sur les applications aide également à ajouter de la sémantique aux archives web, augmentant leur valeur pour les utilisateurs.

Notre travail peut être étendu de plusieurs manières, que nous allons explorer dans des travaux ultérieurs. Tout d'abord, nous pouvons enrichir le langage de motifs que nous utilisons afin d'autoriser des règles de détection et d'extraction complexe, vers un support complet de XPath ou même d'un langage de navigation web plus puissant comme OXPath (Furche *et al.*, 2011), autorisant le crawl d'applications web complexes faisant usage d'AJAX ou de formulaires web. Il y a un compromis, cependant, entre le pouvoir expressif du langage et la simplicité de l'adaptation aux changements de structure. Deuxièmement, nous voulons aller vers une base de connaissances d'applications web construite automatiquement, soit en demandant à un être humain d'annoter les parties d'une application web à extraire ou crawler (Kranzdorf *et al.*, 2012), avec

des techniques d'apprentissage semi-supervisées, soit même en découvrant de manière complètement non supervisée de nouveaux types d'applications web (c'est-à-dire, d'applications web) par une comparaison de la structure de différents sites web, pour déterminer le chemin optimal de crawl par échantillonnage, dans l'esprit de iRobot (Cai *et al*., 2008).

Remerciements

Ce travail a été financé par le septième programme cadre de l'Union européenne (FP7/2007–2013), accord de projet 270239 (ARCOMEM).

Bibliographie

Amitay E., Carmel D., Darlow A., Lempel R., Soffer A. (2003). The connectivity sonar : Detecting site functionality by structural patterns. In *HT*.

ARCOMEM Project. (2011–2013). http://www.arcomem.eu/.

Artail H., Fawaz K. (2008). A fast HTML Web page change detection approach based on hashing and reducing the number of similarity computations. *Data Knowl. Eng.*.

Boser B. E., Guyon I., Vapnik V. (1992). A training algorithm for optimal margin classifiers. In *COLT*.

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

Chakrabarti S., Berg M. van den, Dom B. (1999). Focused crawling : A new approach to topic-specific Web resource discovery. *Computer Networks*, vol. 31, n° 11–16.

Chidlovskii B. (2001). Automatic repairing of Web wrappers. In *WIDM*.

Coleman S. (2008). Blogs and the new politics of listening. *The Political Quarterly*, vol. 76, n° 2.

Diao Y., Altinel M., Franklin M. J., Zhang H., Fischer P. (2003). Path sharing and predicate evaluation for high-performance XML filtering. *ACM TODS*.

Edmonds J. (1967). Optimum branchings. *J. Research Nation Bureau of Standards*, vol. 71B.

Faheem M. (2012). Intelligent crawling of Web applications for Web archiving. In *WWW phd symposium.*

Faheem M., Senellart P. (2013). Intelligent and adaptive crawling of Web applications for Web archiving. In *Icwe.*

Ferrara E., Baumgartner R. (2010). Automatic wrapper adaptation by tree edit distance matching. In *Combinations of intelligent methods and applications.* Springer.

Furche T., Gottlob G., Grasso G., Schallhart C., Sellers A. J. (2011). OXPath : A language for scalable, memory-efficient data extraction from Web applications. *PVLDB*, vol. 4, n° 11.

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

Giles J. (2005). Internet encyclopaedias go head to head. *Nature*, vol. 438.

Gulhane P., Madaan A., Mehta R., Ramamirtham J., Rastogi R., Satpal S. *et al*. (2011). Web-scale information extraction with vertex. In *ICDE.*

Guo Y., Li K., Zhang K., Zhang G. (2006). Board forum crawling : A Web crawling method for Web forums. In *Web Intelligence.*

ISO. (2009). *ISO 28500 :2009, Information and documentation – WARC file format.*

Jupp E. (2012, novembre). Obama's victory tweet 'four more years' makes history. *The Independent*. (http://ind.pn/RF5Q6O)

Kolari P., Finin T., Joshi A. (2006). SVMs for the blogosphere : Blog identification and splog detection. In *AAAI.*

Kranzdorf J., Sellers A. J., Grasso G., Schallhart C., Furche T. (2012). Visual OXPath : robust wrapping by example. In *WWW.* (DÃ©monstration)

Kushmerick N. (1999). Regression testing for wrapper maintenance. In *AAAI.*

Lerman K., Minton S. N., Knoblock C. A. (2003). Wrapper maintenance : A machine learning approach. *J. Artificial Intelligence Research*.

Lim S.-J., Ng Y.-K. (2001). An automated change-detection algorithm for HTML documents based on semantic hierarchies. In *ICDE.*

Lindemann C., Littig L. (2006). Coarse-grained classification of Web sites by their structural properties. In *CIKM.*

Lindemann C., Littig L. (2007). Classifying Web sites. In *WWW.*

Masanès J. (2006). *Web archiving*. Springer.

Meng X., Hu D., Li C. (2003). Schema-guided wrapper maintenance for Web-data extraction. In *WIDM.*

Mulvenon J. C., Chase M. (2002). *You've got dissent ! chinese dissident use of the internet and beijing's counter strategies*. Rand Publishing.

Nielsen M. A. (2011). *Reinventing discovery : The new era of networked science*. Princeton University Press.

Osuna E., Freund R., Girosi F. (1997). An improved training algorithm for support vector machines. In *Workshop on neural networks for signal processing.*

Papailiou N., Konstantinou I., Tsoumakos D., Koziris N. (2012). H2RDF : adaptive query processing on RDF data in the cloud. In *WWW.*

Raposo J., Pan A., Álvarez M., Hidalgo J. (2007). Automatically maintaining wrappers for semi-structured Web sources. *Data Knowl. Eng.*.

Royal Pingdom. (2012). *WordPress completely dominates top 100 blogs.* http://goo.gl/eifRJ.

Sigurðsson K. (2005). Incremental crawling with Heritrix. In *IWAW.*

The Apache Software Foundation. (2012). http://hbase.apache.org/.

The Future Buzz. (2009). *Social media, Web 2.0 and internet stats.* http://goo.gl/H0FNF.

W3C. (1999). *XML path language (XPath) version 1.0.* http://www.w3.org/TR/xpath/.

W3C. (2008). *Extensible markup language (XML) 1.0 (fifth edition).* http://www.w3.org/TR/REC-xml/.

W3C. (2009). *Web application description language.* http://www.w3.org/Submission/wadl/.

WordPress. (2012). *WordPress sites in the world.* http://en.wordpress.com/stats/.

Xia Y., Yang Y., Ge F., Zhang S., Yu H. (2011). Automatic wrappers generation and maintenance. In *PACLIC*.

Ying H.-M., Thing V. (2012). An enhanced intelligent forum crawler. In *CISDA*.

Zheng S., Song R., Wen J.-R., Wu D. (2007). Joint optimization of wrapper generation and template detection. In *KDD*.