Journal of Universal Computer Science, vol. 14, no. 11 (2008), 1857-1876
submitted: 30/9/07, accepted: 25/1/08, appeared: 1/6/08 © J.UCS

Structure-Based Crawling in the Hidden Web

Marcio Vidal, Altigran S. da Silva
Edleno S. de Moura, João M.B. Cavalcanti
(Federal University of Amazonas, Manaus, Brazil
{mvidal,alti,edleno,john}@dcc.ufam.edu.br)

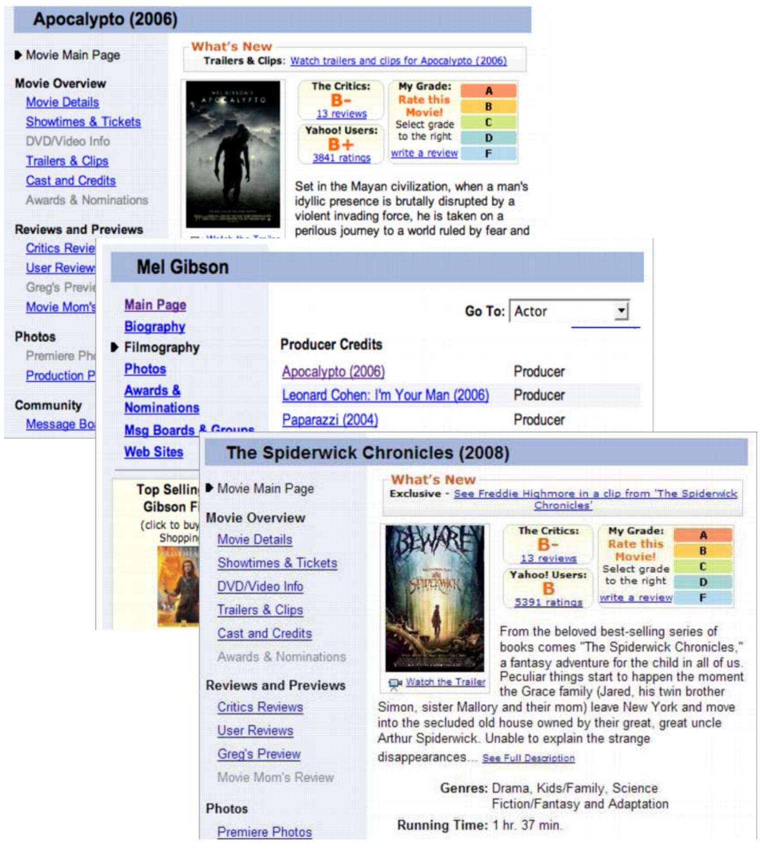
Abstract: The number of applications that need to crawl the Web to gather data is growing at an ever increasing pace. In some cases, the criterion to determine what pages must be included in a collection is based on theirs contents; in others, it would be wiser to use a structure-based criterion. In this article, we present a proposal to build structure-based crawlers that just requires a few examples of the pages to be crawled and an entry point to the target web site. Our crawlers can deal with form-based web sites. Contrarily to other proposals, ours does not require a sample database to fill in the forms, and does not require the user to interact heavily. Our experiments prove that our precision is 100% in seventeen real-world web sites, with both static and dynamic content, and that our recall is 95% in the eleven static web sites examined.

Key Words: Web crawling, hidden web, tree-edit distance, web wrappers
Category: H.3.3, H.3.4, H.3.5, H.3.7

1 Introduction

Crawling web sites to gather collections of pages that contain valuable data is more and more frequent. Common criteria to determine which pages must be included in a collection are usually content-based. There are situations, however, in which the structure of the web pages provides better criteria. For instance, consider an application to collect information about films from the Yahoo! web site. Defining a set of content-related features that encompasses film pages only would be very difficult. For instance, if we train a traditional content-based classifier, it is likely that pages related to actors, theatres or studios would be classified positively. Furthermore, film pages whose content-related features are not related to the training pages, e.g., pages of a different genre, might be classified negatively. For instance, the film page at top of Figure 1 shares content-related features with the artist page in the middle, but it does not share any content-related features with the page at the bottom, despite the fact that they both are film pages. In these cases, using content-related features is likely to fail.

Content-based crawling has been paid much attention, but structure-based crawling techniques have been overlooked. Nevertheless, this is an interesting choice in many cases, e.g., providing web pages to wrappers, which usually rely on structural patterns [Arasu and Garcia-Molina, 2003] [Reis et al., 2004] [Zhai and Liu, 2005] [Chang et al., 2006] [Turmo et al., 2006], structure-aware querying and searching of web pages [Ahnizeret et al., 2004] [Davulcu et al., 1999], public digital libraries on the Web [Calado et al., 2002] [Qin et al., 2004], or web usage mining [Cooley, 200]. This motivated us to work on a web crawling technique that is based on the structure of the



Apocalypto (2006)

▶ Movie Main Page

Movie Overview
Movie Details
Showtimes & Tickets
DVD/Video Info
Trailers & Clips
Cast and Credits
Awards & Nominations

Reviews and Previews
Critics Revie
User Review
Greg's Previe
Movie Mom's

Photos
Premiere Ph
Production P

Community
Message Bo

What's New
Trailers & Clips: Watch trailers and clips for Apocalypto (2006)

The Critics: B- 13 reviews
Yahoo! Users: B+ 3841 ratings
My Grade: Rate this Movie! Select grade to the right write a review
A B C D F

Set in the Mayan civilization, when a man's idyllic presence is brutally disrupted by a violent invading force, he is taken on a perilous journey to a world ruled by fear and

Mel Gibson

Main Page
Biography
▶ Filmography
Photos
Awards & Nominations
Msg Boards & Groups
Web Sites

Go To: Actor

Producer Credits

Apocalypto (2006)	Producer
Leonard Cohen: I'm Your Man (2006)	Producer
Paparazzi (2004)	Producer

Top Sellin
Gibson F
(click to buy
Shoppin

The Spiderwick Chronicles (2008)

▶ Movie Main Page

Movie Overview
Movie Details
Showtimes & Tickets
DVD/Video Info
Trailers & Clips
Cast and Credits
Awards & Nominations

Reviews and Previews
Critics Reviews
User Reviews
Greg's Preview
Movie Mom's Review

Photos
Premiere Photos

What's New
Exclusive - See Freddie Highmore in a clip from 'The Spiderwick Chronicles'

The Critics: B- 13 reviews
Yahoo! Users: B 5391 ratings
My Grade: Rate this Movie! Select grade to the right write a review
A B C D F

Watch the Trailer

From the beloved best-selling series of books comes "The Spiderwick Chronicles," a fantasy adventure for the child in all of us. Peculiar things start to happen the moment the Grace family (Jared, his twin brother Simon, sister Mallory and their mom) leave New York and move into the secluded old house owned by their great, great uncle Arthur Spiderwick. Unable to explain the strange disappearances... See Full Description

Genres: Drama, Kids/Family, Science Fiction/Fantasy and Adaptation

Running Time: 1 hr. 37 min.

Figure 1: Sample pages from the Yahoo! web site.

web pages. Our technique requires an entry point to a web site, and a handful sample pages to be fetched by the user. (More often than not, only one page is enough.) For instance, assume that the entry point is www.yahoo.com, and that the sample page is the one at the top of Figure 1; our technique first traverses the web site looking for pages that are similar to the sample page from a structural point of view, which includes the page at the bottom; then, all of the paths that lead to target pages are used to generate a navigation pattern [Lage et al., 2004], which is composed of links patterns a crawler has to follow to reach the target pages; finally, a crawler based on these patterns is generated automatically. In many cases the crawling process must be performed on pages that are generated on-demand from back-end databases, i.e., pages from the Hidden Web, and they require the crawler to fill in search forms. Our proposal learns the arguments to fill in a form and includes this information in the navigation patterns so that they are available to the final crawler. Addressing the Hidden Web is important since the number of organisations that provide valuable data behind a search form is larger than ever, e.g., department stores, newspapers, research institutions, or media and entertainment companies. A key aspect of our technique is that it can be more precise than other techniques based on content criteria, which seldom achieve high precision levels. Our experiments prove that the structure-based approach tends to be extremely precise; we have performed our experiments on 17 real-world sites, and our crawlers were able to collect almost all the pages that matched the samples, including pages added long after the crawler was generated; we achieved 100% precision and at least 95% recall in the case of static pages.

The rest of the paper is organised as follows: Section 2 reports on some related research initiatives; in Section 3, we present basic concepts related to structural similarity between web pages; in Section 4, we focus on the details of the structure-based crawling approach and the techniques used for generating structure-based web crawlers; Section 5 reports on how we address dynamic web sites; Section 6 reports on the experiments we performed; finally, Section 7 concludes the paper and highlight future research directions.

2 Related Work

We use the structure of the DOM trees that underlie every web page to create clusters of pages that are similar from a structural point of view. This topic was addressed in [Crescenzi et al., 2005] and [Reis et al., 2004], where the goal was to organise a collection of web pages to feed wrappers. Our problem is related, but different since we need to collect web pages that are similar to a few sample pages provided by the user. Our proposal also differs regarding how structural similarity is calculated. In [Crescenzi et al., 2005] the authors rely on some layout properties of links since they use the set of paths between the root of a page and the anchor HTML tags to characterise the structure. Pages that share the same paths are considered structurally similar. Instead, our technique relies on a variation of a tree-edit distance algorithm called RTDM [Reis et al., 2004]. The use of features other than page contents has been explored in [Aggarwal et al., 2001] [Liu et al., 2004], for instance, where features such as the tokens in the URLs and the linking structure of the sites have been examined; however, none of them allows for the internal structure of the web pages.

Our technique relies on using navigation patterns [Lage et al., 2004], which were introduced to guide the navigation of a crawler to fetch pages of interest for data extraction tasks. Our contribution is that the set of navigation patterns is not set beforehand, and they are not hand-crafted, but calculated automatically.

It is also worth mentioning that many current techniques do not allow for dynamic web sites, i.e., sites in which the pages are generated on demand. Some techniques allow the crawler to reach such pages, but require heavy user intervention to fill in search forms [Golgher et al., 2000] [Raghavan and Garcia-Molina, 2001]. In other approaches, a database of common search terms must be created beforehand to help fill in

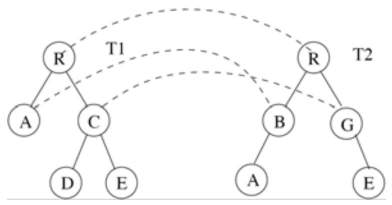


Figure 2: Restrict top-down example.

the forms [Bergmark et al., 2002] [Liu et al., 2004] [Lage et al., 2004]; other proposals fetch pieces of text from the pages that contain forms and use them to fill them in [Hedley et al., 2004].

3 Structural Similarity

Note that the structure of a web page can be described by a DOM tree, which is a labelled ordered rooted tree. (A rooted tree is a tree whose root vertex is fixed; ordered rooted trees are rooted trees in which the relative order of the children is fixed for each vertex; labelled ordered rooted trees have a label attached to each of their vertices.) In our proposal, structural similarity between two pages is based on the tree-edit distance between their corresponding DOM trees [Selkow, 1977]. Intuitively, the edit distance between trees T_1 and T_2 is the cost of the minimal set of operations required to transform T_1 into T_2, i.e., the similarity between T_1 and T_2 is inverse to their edit distance. We take three operations into account, namely: vertex removal, vertex insertion, and vertex replacement, each with a cost. We use a restricted top-down version of the tree-edit distance problem which restricts the removal and insertion operations to take place in the leaves of the trees only [Chawavate, 1999], which is particularly useful to deal with DOM trees. This version is referred to as RTDM [Reis et al., 2004].

Let T_1 and T_2 be to trees with n_1 and n_2 vertices, respectively. To determine the restricted top-down tree-edit distance between them, the RTDM algorithm first finds all identical sub-trees that occur at the same level, which is performed in $O(n_1)$ time using a graph of equivalence classes. Then, an adaptation of Yang's algorithm [Yang, 1991] is applied to obtain the minimal restricted top-down mapping between the trees, which is performed in $O(n_1 n_2)$ worst-case time. (In practice, it performs better since only a restricted set of operations can be applied). The results of the algorithm are illustrated in Figure 2, where dashed lines denote node mappings.

4 Generating Crawlers

We use the Yahoo! films web site to illustrate our ideas. This site provides information about films, actors, directors, producers, and so on, but we are interested in film pages

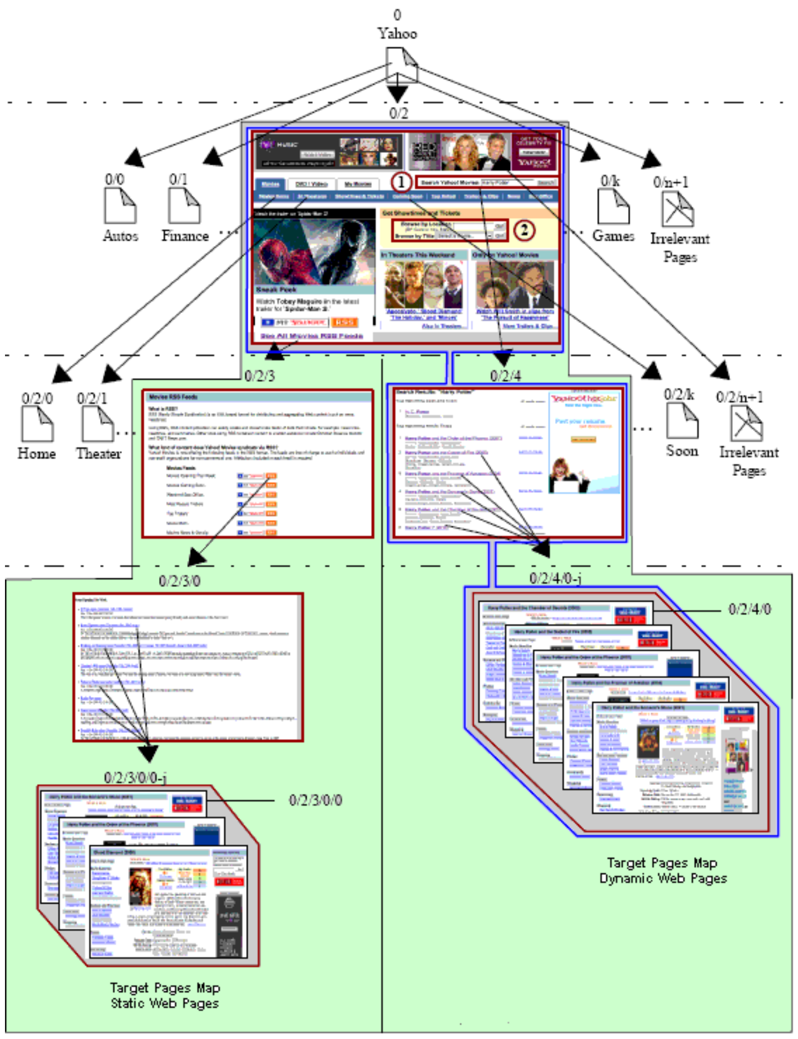


Figure 3: Overall structure of the Yahoo! film web site.

only. The overall structure of this site is sketched in Figure 3. In this section, we focus on the static part of the site, i.e., pages with prefix $0/2/3$ on the left of the figure. The dynamic part, i.e., pages with prefix $0/2/4$, will be addressed in Section 5.

Assume that the pages with a label of the form $0/2/3/i$ $(1 \leq i \leq k)$ are similar to the page labelled $0/2/3/0$, which has links to film pages. We call the pages that are referenced to by these links film hub pages. Furthermore, assume that the pages with labels of the form $0/2/3/i/j$ $(1 \leq i \leq K, 1 \leq j \leq K_i)$ are similar to page $0/2/3/0/0$, i.e., they all are film pages. A crawler for this site would start at page 0; it then would

Id	Regular Expression	Applied to
E_0	http://www.yahoo.com	Entry Point
E_1	http://films.yahoo.com/	First level
E_2	http://films.yahoo.com/rss	Second level
E_3	http://rss.ent.yahoo.com/films/ˆ[a-zA-Z]+$.xml	Films Hub Page
E_4	http://films.yahoo.com/shop?d=hv&cf=info&id=ˆd+$&intl=us	Film Page

Table 1: Example of a simple navigation pattern using Perl-like syntax.

have access to all of the film hub pages and would be able to fetch all of the film pages. Notice that new film pages can be added without a notice, which implies that the film hub pages may be updated to include new links to new film pages. Obviously, these new links and pages need to be fetched in future crawls. Our technique requires two input parameters: a URL that serves as an entry point to the site to be crawled, and another URL to a sample web page that is representative enough of the pages to be fetched. In our example, page 0 is the entry point, and page $0/2/3/0/0$ serves as the sample page.

Generating a crawler comprises two phases: site mapping and navigation pattern generation. In the former, all of paths starting from the entry point are traversed looking for target pages. A page is considered to be a target if it is structurally similar to the given sample page. Notice that, the process is limited to a web site, i.e., no path to external sites is processed. Every path from the entry point that leads to a target page is recorded, i.e., the output of the site mapping phase is a minimum spanning tree in which all leaves are target pages, and it is referred to as the target pages map. In the latter phase, the goal is to create a generic representation of the target pages map that consists of a list of regular expressions that represent the links in a page that the crawler has to follow to reach the target pages. This generic representation is called a navigation pattern. Note that several paths to the target pages may exist; in such cases, we choose the one that potentially leads to the largest number of target pages.

Table 1 shows a real-world navigation pattern for the Yahoo! web site: E_0 corresponds to the entry page, E_1 corresponds to the first level, E_2 matches the links to film RSS feeds, E_3 matches film hub pages only, and E_4 is applied to each RSS feed hub page and matches every link that leads to a film page. Notice that no user intervention other than providing the entry point and the sample page is required.

4.1 Site Mapping

The goal of this phase is to generate a target pages map, for which we simply crawl the web site starting from the entry point using a breadth-first traversal procedure. (The breadth-first traversal is the best choice for mapping the target web site, since this strategy provides a better coverage of the sites traversed due to the fact that more relevant

Algorithm 1 Procedure used for the site mapping phase.

```
1: function MAPPING($e, p$)          ▷ $e$ is an entry point to a site, and $p$ is a sample page
2:     $Q \leftarrow$ a queue with the links extracted from $e$
3:     $X \leftarrow \emptyset$
4:     while $Q \neq \emptyset \wedge \neg$STOPPING do
5:         $x \leftarrow dequeue(Q)$
6:         if RTDMSIM($p, x$) then
7:             $X \leftarrow X \cup \{x\}$
8:         else
9:             $Q \leftarrow Q \cup \{$links extracted from $x\}$
10:        end if
11:    end while
12:    Let $T \leftarrow$ an empty target pages map
13:    for all $x \in X$ do
14:        Let $C = \langle e, v_1, \ldots, v_k, x\rangle$ be the path from $e$ to $x$
15:        Add the nodes and links in $C$ to $T$
16:    end for
17:    return $T$                    ▷ Return the target pages map in $T$
18: end function
```

pages are usually found in upper levels of web sites [Najork and Wiener, 2001].) Below, we provide a formal definition of a target pages map:

Definition 1. Let $S = \langle P, L\rangle$ be a web site, where P is a set of pages and L is a set of pairs $\langle x, y\rangle$ such that a link exists from page x to page y; let $e, p \in S$ be the entry point and a sample page supplied by the user, respectively. We define a target pages map as a tree T that is a subset of the minimum spanning tree of S in which e is the root and the set of leaves is consists of the set of nodes p_i that are structurally similar to p.

A formal definition of our procedure is presented in Algorithm 1. In line 2, all links in e are queued in Q; next, the crawling process iterates over this queue in the loop in lines 4–11. This loop ends when the queue is empty or another STOPPING condition is met, e.g., the maximum number of pages to be fetched or the maximum number of levels to be explored are reached. In our current implementation, the STOPPING condition is set by means of user-defined parameters, which are fairly simple to be estimated. Notice that these parameters are used only for the site mapping phase and are not required once the resulting crawler is generated. In lines 5 and 6, the first element in the queue is compared with p using the RTDMSIM function. This comparison, is based on the structure of these pages, i.e., we verify how similar their underlying DOM trees are using the RTDM algorithm, cf. Section 3. If the current page x and the sample page p are considered structurally similar, we add x to a set X of target pages in line 7; otherwise, the links in x are enqueued in line 9, and the crawling process goes on. At the end of each iteration, set X has all target pages found. In the loop in lines 13–16, every node and link in the paths from the entry point to target pages in X is be added to the target pages map T. To avoid links to be visited more than once, the *dequeue* operation simply marks the link on the top as visited, instead of actually removing it

Algorithm 2 Procedure used for grouping nodes.

```
 1: function GROUP($r$)                                              ▷ $r$ is a node
 2:     Let $G = \{G_1, G_2, \ldots, G_k\}, k \leq |\mathcal{C}(r)|$ such that     ▷ $\mu(x)$ denotes the URL of node $x$
         $G_1 \cup \ldots \cup G_k = \mathcal{C}(r)$,                 ▷ $\mathcal{C}(x)$ denotes the set of children of node $x$
         $G_i \cap G_j = \emptyset$, and
         $c_\ell \in G_i$ iff $c_m \in G_i \wedge$ URLSIM$(\mu(c_\ell), \mu(c_m))$
 3:     for all $G_i = \{c_{i_1} \ldots c_{i_{k_i}}\}$ do
 4:         $\pi \leftarrow$ URLPATTERN$(\mu(c_{i_1}) \ldots \mu(c_{i_{k_i}}))$
 5:         $n_i \leftarrow \langle \pi, c_{i_1} \cup \ldots \cup c_{i_{k_i}} \rangle$
 6:         Remove all $c_{i_j} \in G_i$ as a child of $r$
 7:         Add $n_i$ as a child of $r$
 8:         Call GROUP($n_i$)
 9:     end for
10: end function
```

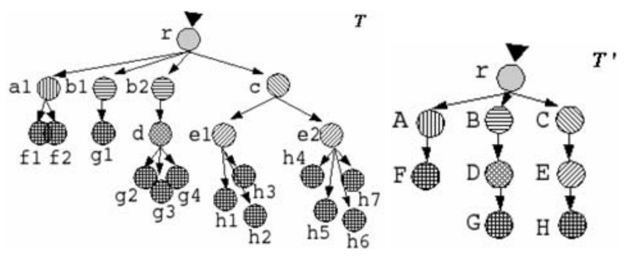
from the queue, and moves the top pointer to the next link, if any; furthermore, a link is not queued if it is already on the queue, even if it is marked as visited.

4.2 Navigation Pattern Generation

The goal of this phase is to build a navigation pattern from the target pages map generated in the mapping phase. Such pattern consists of a list of regular expressions that represent the links in a page the crawler has to follow to reach the target pages. In order to generate it, we have to generalise the URLs of the pages in the paths of the target pages map using regular expressions, and select the best paths that lead to the desired target pages amongst all of the paths from the entry point.

The first step consists of grouping the nodes in the target pages map tree that represent pages whose URLs are considered similar, cf. Section 4.3. The procedure for performing this grouping is presented in Algorithm 2. It is applied to the root of the target pages map tree initially, and it is then applied recursively to its children nodes. For a given node r, line 4 defines a partitioning on the set of children of r, where each partition G_i only contains nodes that correspond to pages with similar URLs. In lines 6–9, a new node n_i is created from the nodes in G_i as follows: the URL in n_i corresponds to a regular expression π that generalises the set of URL from the nodes in G_i according to function URLPATTERN, cf. Section 4.3. The set of children of n_i is composed of the children of the nodes in G_i. Then, the nodes in G_i are removed from the set of children of r in line 8, and replaced by node n_i in line 9. In summary, we replace all nodes corresponding to pages with similar URLs by a single node that represents them. In line 10, the GROUP procedure is called recursively to deal with the new node n_i.

The execution of the GROUP procedure is illustrated in Figure 4, in which the left side sketches a target pages map T, and the right side illustrates the tree T' returned by this procedure. (Nodes with the same fill pattern correspond to pages with similar URLs.) Each node represented by a capital letter in T', except for the root r, groups all



(a) (b)

Figure 4: Sample execution of the GROUP procedure.

nodes from T represented by non capital letters. Notice that nodes labelled A, C and D actually represent single nodes, a1, c and d, respectively.

This procedure may lead to a navigation pattern in which there is more than one path that leads to the target pages. This is depicted in Figure 4, in which three paths are found: r to F, r to G, and r to H. From these paths, we choose the one that leads to the greatest number of target pages. In our example, path r to H satisfies this heuristic. This notion is formalised in the following definition:

Definition 2. Let S be a web site, and let T be a target pages map for S obtained when $e, p \in S$ are the entry point and the sample page supplied by the user, respectively. A navigation pattern is the path in the tree T' obtained after applying the GROUP procedure on T, which begins with e and ends at the node of T' that corresponds to the greatest number of target pages in T.

4.3 Evaluating Similarity between URLs

We consider that a URL is a string formed by several substrings separated by a "/" that are referred to as levels. Let $u = u_1/u_2/\ldots/u_n$ $(n \geq 1)$ and $v = v_1/v_2/\ldots/v_m$ $(m \geq 1)$ be two URLs; they are considered similar if: they have the same number of levels, i.e., $n = m$; the first level in u and v are equal, i.e., $u_1 = v_1$; and there are at most K levels in the same position in u and v (except for the first one) that are not equal, i.e., if $\delta = \{\langle u_i, v_i\rangle \mid u_i \neq v_i\}$, then $|\delta| \leq K$. By performing preliminary experiments with several actual URLs, we have found that $K = 2$ results in an accurate and safe similarity evaluation.

4.4 Generating URL Patterns

We now report on the procedure used to generate a pattern that represents a set of URLs. Recall that the URLs are assumed to be similar and that they differ at most in K levels. Our strategy for generating URL patterns consists in building regular expressions that

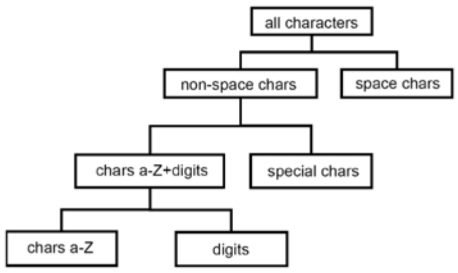


Figure 5: A sample regex tree.

match the strings occurring at levels in which the URLs differ from each other. This requires to introduce the notion of a regex tree.

Definition 3. A regex tree R is a tree in which each node n is associated with a regular expression e_n over an alphabet Σ that denotes a language L_n, such that for each internal node a in R whose children are $\{c_1, \ldots, c_k\}$, it holds that $L_{c_1} \cup \ldots \cup L_{c_k} = L_a$ and $L_{c_1} \cap \ldots \cap L_{c_k} = \emptyset$.

Figure 5 illustrates the kind of regex trees we use, which are adapted from the so-called delimiters tree in [Reis et al., 2002]. According to Definition 3, the regex tree is built so that each node defines a language that is disjoint with the languages in its sibling nodes. In particular, each leaf node defines a language that is disjoint with the languages in its sibling leaf nodes. Thus, when applied to the tokens in a URL, a given token will match exactly one leaf node. This property allows to use the leaf nodes in a regex tree to tokenise a URL or a URL level. Furthermore, each internal node defines a language that is formed by the union of the languages of its children nodes. This property allows to find a single regular expression that matches every token that occurs in a set of URLs at the same position. Let l_t and l_s be two leaf nodes matched by two distinct tokens t and s; the node a that is the deepest common ancestor of l_t and l_s defines a regular expression that matches both t and s.

The complete procedure for generating a URL pattern is described in Algorithm 3. To explain it, we use the set of sample URLs in Figure 6. The URLPATTERN procedure iterates in the loop of lines 4–15 through the levels in the URLs that are different from each other. As illustrated in Figure 6(a), only levels 6 and 7 will be processed in this loop. We focus on level 7, since it is more illustrative of the procedure. This level is detailed in Figure 6(b). According to line 5 there is no common prefix between all $u_i[7]$, but there is common suffix .html between them. In the loop of lines 6–8, we tokenise each infix $f_i[7]$ and generate the tokens in Figure 6(b). Next, in the loop in lines 9–13, each set of tokens in the same position of the infixes is given as an argument for calling the TOKENREGEX procedure. This means that in Figure 6(b), each column labelled $\hat{f}_i[7][j]$ is passed as an argument to TOKENREGEX. The result of each call is shown in

Algorithm 3 Procedure for generating a URL pattern.

```
1: function URLPATTERN($U$)    ▷ $U = \{u_1, u_2, \ldots, u_n\}$ is a set of URLs
2:     $u_\pi \leftarrow u_1$
3:     Let $u_i[k]$ be the $k-$th level of URL $u_i$
4:     for all $k$ such that $\{\langle u_1[k], \ldots, u_n[k]\rangle \mid u_i[k] \neq u_j[k]$ for some $1 \leq i, j \leq n$ do
5:         Let $u_i[k] = p \bullet f_i[k] \bullet s$ where $p(s)$ is the common prefix (suffix) of every $u_i[k]$
6:         for $i \leftarrow 1$ to $n$ do    ▷ Symbol "●" is denotes string catenation
7:             $\langle f_i[k][1], f_i[k][2], \ldots, f_i[k][m_{i,k}]\rangle \leftarrow$ TOKENISE($f_i[k]$)
8:         end for
9:         for $j \leftarrow 1$ to $\max\{m_{1,k}, m_{2,k}, \ldots, m_{n,k}\}$ do
10:            $x \leftarrow$ TOKENREGEX($\hat{f}_1[k][j], \hat{f}_2[k][j], \ldots, \hat{f}_n[k][j]$)
11:                where $\hat{f}_i[k][j] = f_1[k][j]$ if $j \leq m_{k,i}$ or $tf_1[k][j] =$ "" otherwise
12:            $\pi \leftarrow \pi \bullet x$
13:        end for
14:        $u_\pi[k] \leftarrow s \bullet \pi \bullet p$
15:    end for
16:    return $u_\pi$    ▷ Returns a URL pattern
17: end function
18:
19: function TOKENISE($s$)    ▷ $s$ is a string
20:    $R \leftarrow$ an empty regex tree
21:    $i \leftarrow 0$
22:    while $s \neq$ "" do
23:        $i \leftarrow i + 1$
24:        $s_i \leftarrow$ the leftmost largest substring from $s$ that matches a leaf in $R$
25:        $s \leftarrow s - s_i$
26:    end while
27:    return $\{s_1, s_2, \ldots, s_n\}$    ▷ Returns a set of tokens
28: end function
29:
30: function TOKENREGEX($T$)    ▷ $T = \{t_1, t_2, \ldots, t_n\}$ is a set of tokens
31:    if $t_i = t_j$ for all $t_i, t_j \neq$ "" then
32:        $p_T \leftarrow t_i$
33:    else
34:        Let $\eta(t_i)$ be the leaf node that matches token $t_i \neq$ ""
35:        $a \leftarrow$ the deepest an ancestor of all $\eta(t_i)$
36:        $p_T \leftarrow e_a$
37:    end if
38:    if there is some $t_i =$ "" then
39:        $p_T \leftarrow p_t \bullet$ "$*$"
40:    end if
41:    return $p_T$    ▷ $p_T$ is a regex that matches the tokens in $T$
42: end function
```

the line labelled with π. For column $\hat{f}_i[7][1]$, all of the tokens match the same leaf in the regex tree in Figure 5. Thus, the regular expression in this node is used. For column $\hat{f}_i[7][2]$, since the same token appears in every line, it is simply considered as the regular expression. In the case of the tokens in column $\hat{f}_i[7][5]$, notice that tokens 8 and D match distinct leafs in the regex tree and the deepest common ancestor corresponds to the node whose regular expression is \w. Thus, this regular expression is used.

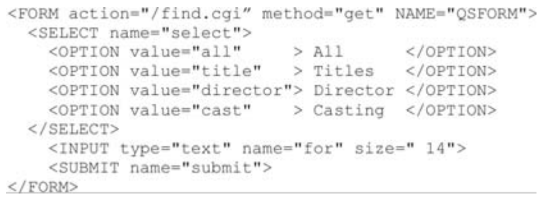
	$u_i[1]$	$u_i[2]$	$u_i[3]$	$u_i[4]$	$u_i[5]$	$u_i[6]$	$u_i[7]$
u_1	informatik.uni-trier.de	∼ley	db	indices	a-tree	c	Chaves:Silvio_8=.html
u_2	informatik.uni-trier.de	∼ley	db	indices	a-tree	u	Ugher:Mangabeira_D=.html
u_3	informatik.uni-trier.de	∼ley	db	indices	a-tree	m	Mendes:Sergio.html
Π	informatik.uni-trier.de	∼ley	db	indices	a-tree	[a-Z]+	[a-Z]+:[a-Z]+_*\w*.html

(a)

	$\hat{f}_i[7][1]$	$\hat{f}_i[7][2]$	$\hat{f}_i[7][3]$	$\hat{f}_i[7][4]$	$\hat{f}_i[7][5]$	s
$u_1[7]$	Chaves	:	Silvio	_	8	.html
$u_2[7]$	Ugher	:	Mangabeira	_	D	.html
$u_3[7]$	Mendes	:	Sergio	“”	“”	.html
π	$[a-Z]+$	:	$[a-Z]+$	_*	\w*	.html

(b)

Figure 6: Example of a simple navigation pattern using Perl-like syntax.



```
<FORM action="/find.cgi" method="get" NAME="QSFORM">
  <SELECT name="select">
    <OPTION value="all"      > All      </OPTION>
    <OPTION value="title"    > Titles   </OPTION>
    <OPTION value="director"> Director </OPTION>
    <OPTION value="cast"     > Casting  </OPTION>
  </SELECT>
    <INPUT type="text" name="for" size=" 14">
    <SUBMIT name="submit">
</FORM>
```

Figure 7: Sample HTML form.

5 Dealing with Dynamic Web sites

Currently, the number of pages in the hidden web is enormous, which argues for a good crawler to be able to fetch those pages. If a page with a form is found during the mapping phase, it is necessary to check if it can generate new target pages, in which case, it might well be included in the target pages map along with the required parameters to fill it in. In this section, we report on how we deal with dynamic sites.

5.1 Characteristics of HTML Forms

For our purposes, a form can be viewed as the HTML code contained within tags ⟨FORM⟩ and ⟨/FORM⟩. For instance, Figure 7 sketches a form that collects pages about films; it has a menu between tags ⟨SELECT name='select'⟩ and ⟨/SELECT⟩, a text box encoded in tag ⟨INPUT type='text' name='for' size='14'⟩, and the information required to submit the form in tag ⟨FORM action='/find.cgi' method='get' NAME='QSFORM'⟩.

We model a form F as a set $\{(t_1, v_1), (t_2, v_2), \ldots, (t_k, v_k)\}$ in which each pair (t_i, v_i) corresponds to a field f_i with type t_i and (possibly) value v_i. The type of a field denotes if it is a selection list, a text box, a check box, or a radio button, for instance. (If t_i indicates a text box, then $v_i = text$.) For instance, the form in Figure 7 is modelled as follows: $F = \{\langle \text{select, all} \rangle, \langle \text{select, title} \rangle, \langle \text{select, director} \rangle, \langle \text{select, cast} \rangle, \langle \text{for, text} \rangle\}$.

Regarding the submission method, it can be either GET or POST. The main difference between them is the way the browser encodes the data sent to the Web server. Whereas using GET the submission is through the URL, using POST encodes the data into the HTTP header. With the GET method, queries are generated as a concatenation of the following: the URL of the page that has the form; the value defined for the attribute action; the special character ?; t_1, that is, the name of the first input type of the form; the symbol "="; and the value v_1; if there are other fields in the form, they are encoded and appended to the URL using symbol "&" as a separator. In tag ⟨FORM⟩ in Figure 7, the value of the attribute action is find.cgi. If the URL of the page with this form is http://www.myhtmlform.com, the following are then samples of the URLs to submit this form using the GET method: http://www.myhtmlform.com/find.cgi?select=all&for=mytext, http://www.myhtmlform.com/find.cgi?select=title&for=mytext.

5.2 Automatic Web Form Filling

Our technique first verifies if the form requires parameters like e-mail addresses, user names or passwords, or if it relies on Java script, in which case we consider the form impossible to handle. If it can be handled, we extract the fields and create the set of pairs $\langle type, value \rangle$; for instance, in Figure 7, type select has four values, namely: all, title, director, and cast. For forms that do not have text fields, all possible combinations of types and values are used to create a query vector. However, if the form has one text field at least, our technique uses search terms that are sampled from the sample and the form pages themselves, and it then fills in the form and gathers the corresponding answer pages. We focus on the less frequent terms, and then we extract the labels of the text fields and generate a set P' of pairs $\langle label, value \rangle$. We have adapted the well-know heuristics for label extraction used in [Fontes and Silva, 2004], [Lage et al., 2004] and [Zhang et al., 2004]. Contrarily to [Chakrabarti et al., 2002], [Bergmark et al., 2002], [Lage et al., 2004] and [Liu et al., 2004], we do not use a predefined database to fill in text fields; instead, we consider only the 10 less frequent terms in the sample page and in the form page. This value is based on the results of our extensive tuning experiments.

For instance, consider the form in Figure 7 and the set of the 10 less frequent terms occurring in the given sample page and in the form page, namely: Harry, Potter, Joanne, Kathleen, Rowling, philosopher, quidditch, Voldemort, Hogwarts, and Azkaban. The following queries would be generated amongst the possible queries for this form: select=all&for=Harry, select=all&for=Potter, select=all&for=Joanne, and select=title&for=Harry. Once we have all the possible combinations generated for each form field, we proceed to the submission phase for collecting the answer pages.

5.3 Submission Plan and Crawling Answers Pages

Consider a set of queries, constructed as discussed in the previous section. Initially, all queries are submitted and the corresponding answer pages are collected. An analysis of the answer pages is necessary to detect useless pages, e.g., pages with error messages or empty pages. We consider useful the 30% largest pages only (size in bytes), which is based on our extensive tuning experiments. There are several cases regarding the answer pages thus collected:

Target Pages: If the page is structurally similar to the sample page, it is stored, and the form page being processed is added to the target pages map.

Form pages: To deal with new forms, the procedure described in Section 5.2 is applied recursively.

Pages with links: This is the most common case. A naive approach to deal with these pages is to follow each link in the usual way, cf. Section 4; however, our experiments prove that the number of links is usually very large, which argues for grouping them to reduce the total number of links to explore. (Each group has URLs that are similar to each other, cf. Section 4.2.) After that, we follow each link in that group until we find a target page, and add it to the target pages map.

Error pages: If a query is invalid, the result is usually an error page. Proposals such as [Doorenbos et al., 199] and [Barbosa and Freire, 2004] classify a page as an error if it matches pre-defined error templates, e.g., pages with words such as ERROR, Internal Server Error, or 400 Bad Request. In our approach, we simply assume that 30% of the largest pages are originated from valid queries. For this reason, we do not have a specific procedure for cases in which the pages returned are error pages. In exceptional cases, if a form always returns error pages, the processing may continue following any of the links found in these pages. If the list of links extracted from these pages is empty and no target page can be reached, then the form that generated these queries is discarded.

5.4 Incorporating Forms into a Target Pages Map

After the queries are submitted and the corresponding answer pages are analysed, the result is a bunch of queries that lead to target pages from a given form, e.g., http://www.myhtmlform.com/find.cgi?select=all&for=Harry, or http://www.myhtmlform.com/find.cgi?select=title&for=Potter. In turn, these pages may lead to additional forms that can be incorporated into the target pages map if they are processed successfully. This is accomplished by means of a generic URL that encodes the structure of the queries and the parameters they require for execution, e.g., http://www.myhtmlform.com/find.cgi?select=*&for=*. This URL will be incorporated into the navigation pattern and ultimately used by the crawler, as if it was an ordinary node in the target pages map. Notice

Site	Static Sites Description	Target Page
www.ejazz.com.br	Jazz styles, artists, instruments, and so on.	Artists
informatik.uni-trier.de/~ley/	VLDB conferences at DBLP	VLDB Conferences
www.olympic.org	International Olympic Committee	Olympic Heroes
www1.folha.uol.com.br/folha/turism	Travel Site	News
www1.folha.uol.com.br/folha/money	Economics Site	News
www.dot.kde.org	KDE Software Releases	Package Releases
www.amazon.com	Amazon Essential CDs section	CD
www.wallstreetandtech.com	Wall Street Technology	News
www.cnn.com/weather	Wether in the World	Wether Forecast
sports.yahoo.com	Yahoo! sports section	European soccer league
www.nasa.gov/home	NASA web site	News

Site	Dynamic Sites Description	Target Page
www.bestwebbuys.com	On-line products comparison	Harry Potter Books
www.chapters.indigo.ca	On-line store	“Right Now” CDs
www.dvdempire.com	On-line DVD store	“Final Fantasy” DVDs
www.gracenote.com	On-line CD store	CDs by “Eminem Show”
www.talkingbooks.com	On-line Book store	Books on travel
www.zevelekakis.gr	Medical On-line Bookstore	Books on “Heart Disease”

Table 2: List of web sites used in the experiments.

that it is impossible to define, prior to executing the crawler, which parameters must be used for the forms it is expected to find. It is the user who must provide the appropriate arguments for these parameters during the execution of the crawler.

6 Experiments

In our experiments, we used 11 real-world web sites that are well-known to have large collections of data-rich pages. Regarding dynamic sites, we used 6 web sites from the TEL-8 collection [Chang et al., 2003]. These sites have been used in references such as [Zhang et al., 2004] and [Davulcu et al., 1999]. Table 2 provides a little information on these sites.

The experiments were carried out on a Linux-based PC (distribution Gentoo Kernel 2.4), with 1GB RAM, 2.8GHz processor and 80GB HD, and the prototype was implemented in Java. Our experiments consisted of first generating a navigation pattern using a sample page and an entry point, then generating a crawler to process the web site, and, finally, collecting the target pages according to the procedures described earlier.

6.1 Results regarding Static Web Sites

Table 3 shows the results of our experiments on 11 static web sites. Column Inspection reports on the number of pages a person found in each site using a standard browser;

	Target Pages		**Traversed Links**	
Site	**Inspection**	**Automatic**	**Mapping**	**Crawling**
E-jazz	149	149 (100%)	2213	199
VLDB	30	30 (100%)	70	32
OLYMPIC	335	328 (98%)	395	379
Travelling	301	301 (100%)	348	335
Money	470	468 (99%)	550	528
KDE	30	30 (100%)	120	31
CDs	416	398 (96%)	440	426
Wall Street	261	253 (97%)	1579	283
CNN	51	49 (96%)	485	65
Yahoo! Sports	38	37 (97%)	1307	45
NASA	339	325 (95%)	687	389

Table 3: Results of the experiments with static sites.

		Target Pages		**Links**	**New Target**
Site	**Run**	**Inspection**	**Automatic**	**Traversed**	**Pages**
Trip	1	314	308 (98%)	335	7
	2	303	291 (96%)	310	11
Money	1	486	478 (98%)	497	84
	2	482	476 (99%)	497	80
KDE	1	29	29 (100%)	31	14
	2	29	29 (100%)	34	19
CDs	1	409	394 (96%)	492	4
	2	418	412 (98%)	487	18
WallStreet	1	267	257 (96%)	271	17
	2	272	267 (98%)	273	25
NASA	1	334	320 (95%)	339	12
	2	337	323(95%)	341	13

Table 4: Results after new target pages are created.

contrarily, column Automatic reports on the number of pages our crawler found automatically. Note that we reach 100% precision in all of the cases, which implies that we do not collect any inadequate page. Column Traversed Links reports on the number of links covered to generate the crawler during the mapping and the crawling phases, respectively. Notice that the figures in the latter column are smaller than the corresponding figures in the former column; this happens because the only links that match

Site	Parameter	Value
www.bestwebbuys.com	searchfor	title
	title	Harry Potter
www.chapters.indigo.ca	section	music
	keyword	Right Now
www.dvdempire.com	search_type	title
	media	DVD
	search_string	Final Fantasy
www.gracenote.com	q_artist	Eminem Show
www.talkingbooks.com	search_by	title
	search_for	Health
www.zevelekakis.gr	title	Heart Disease

Table 5: Parameters and values used with dynamic web sites.

		Automatic Crawling	
Site	Inspection	Training	Production
www.bestwebbuys.com	334	240	330 (98%)
www.chapters.indigo.ca	13	140	13 (100%)
www.dvdempire.com	14	80	14 (100%)
www.gracenote.com	20	15	20 (100%)
www.talkingbooks.com	23	80	23 (100%)
www.zevelekakis.gr	282	15	280 (99%)

Table 6: Results of the experiments with dynamic sites.

the regular expressions in the navigation pattern are followed. In our experiments, we executed the crawlers twice. In some cases, we detected that new pages were added to the corresponding site since the first run, cf. Table 4. Notice that the figures in columns Target Pages and Links Traversed are similar in both Tables 3 and 4.

6.2 Results regarding Dynamic Web Sites

We conducted our experiments on six dynamic web sites, cf. Figure 6, and the form fields were filled in using the specification in Table 5. Again, column Inspection in Table 6 correspond to the number of pages a person found using a standard browser. The figures in column Training correspond to the number of pages collected from the forms in each site. Notice that, the answer pages were always target pages in the six sites under examination. The results are presented in column Production. Notice that we achieved 100% precision, i.e., no page other than target pages was collected; the percentage in this column corresponds to the ratio of pages collected by a person with regard to the number pages collected automatically. This percentage can be interpreted as the level of recall reached by the crawlers in dynamic web sites. Notice that the usual notion of recall does not make sense for dynamic web sites because the number of pages to collect depends on the value of the arguments used to fill in the forms.

7 Conclusion and Future Work

In this article we have proposed and evaluated a new approach for generating structure-based web crawlers automatically. This new approach uses the structure of web pages instead of their content to determine which pages should be collected. Our method also deals with dynamic web sites, since our crawlers are able to fill in a web form with minimal user intervention. Our experiments indicate that the new structure-based approach can be extremely effective, since it results in high precision and recall levels. Furthermore, our proposal requires only a few examples to learn how to identify the set of target pages; in reality, we have used just one example in our experiments, whereas it is usually necessary to provide a few dozen using other well-known content-based techniques, cf. [Chakrabarti et al., 2002] [Liu et al., 2004] [Qin et al., 2004]. The structure-based approach is complementary to the traditional content-based approach, in the sense that it is more suited for sites that are both data intensive and regular. This means that our new method is the best option for a restricted set of crawling tasks.

In future, we wish to extend our method so that the resulting crawlers are able to retrieve pages from several distinct web sites. For this, we plan to combine the content-based and the structure-based approaches, i.e., a hybrid strategy. The idea is to produce methods that are more flexible than the structure-based method proposed here, while still achieving high precision and recall levels.

Acknowledgments

This work was partially supported by projects GERINDO (CNPq/CT-INFO 552.087/02-5), SIRIAA (CNPq/CT-Amazônia 55.3126/05-9), 5S-VQ (CNPq/CT-INFO 55.1013/05-2), UOL through its “UOL Bolsa Pesquisa” program (Proc. Num. 0503301456), and grants by CNPq to A.S. da Silva (308528/07-7), E.S. de Moura (302209/07-7), J.M.B. Cavalcanti (303738/06-5), and a grant by FAPEAM to M. Vidal.

References

[Aggarwal et al., 2001] Aggarwal, C., Al-Garawi, F., and Yu, P. (2001). On the design of a learning crawler for topical resource discover. *Transactions on Information System*, 19(3):286–309.

[Ahnizeret et al., 2004] Ahnizeret, K., Fernandes, D., Cavalcanti, J., de Moura, E., and da Silva, A. (2004). Information retrieval aware web site modelling and generation. In *Proceedings of the 23rd Conference on Conceptual Modeling*, pages 402–419.

[Arasu and Garcia-Molina, 2003] Arasu, A. and Garcia-Molina, H. (2003). Extracting structured data from web pages. In *Proceedings of the 19th Conference on Management of Data*, pages 337–348.

[Barbosa and Freire, 2004] Barbosa, L. and Freire, J. (2004). Siphoning hidden-web data through a keyword-based interface. In *Anais do XIX Simpósio Brasileiro de Banco de Dados*, pages 309–321.

[Bergmark et al., 2002] Bergmark, D., Lagoze, C., and Sbityakov, A. (2002). Focused crawls, tunneling, and digital libraries. In *Proceedings of the 6th European Conference on Digital Libraries*, pages 91–106.

[Calado et al., 2002] Calado, P., da Silva, A., Ribeiro-Neto, B., Laender, A., Lage, J., Reis, D., Roberto, P., Vieira, M., Gonçalves, M., and Fox, E. (2002). Web-DL, an experience in building digital libraries from the Web. In *Proceedings of the 11th Conference on Information and Knowledge Management*, pages 675–677.

[Chakrabarti et al., 2002] Chakrabarti, S., Punera, K., and Subramanyam, M. (2002). Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th World Wide Web Conference*, pages 148–159.

[Chang et al., 2006] Chang, C.-H., Kayed, M., Girgis, M., and Shaalan, K. (2006). A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428.

[Chang et al., 2003] Chang, K.-C., He, B., Li, C., and Zhang, Z. (2003). Tel-8 query interface. Available at http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/.

[Chawavate, 1999] Chawavate, S. (1999). Comparing hierarchical data in external memory. In *Proceedings of the 25th Conference on Very Large Data Bases*, pages 90–101.

[Cooley, 200] Cooley, R. (200). The use of web structure and content to identify subjectively interesting web usage patterns. *Transactions on Internet Technology*, 3(2):93–116.

[Crescenzi et al., 2005] Crescenzi, V., Merialdo, P., and Missier, P. (2005). Clustering web pages based on their structure. *Data an Knowledge Engineering*, 54(3):277–393.

[Davulcu et al., 1999] Davulcu, H., Freire, J., Kifer, M., and Ramakrishnan, I. (1999). A layered architecture for querying dynamic web content. In *Proceedings of the 25th Conference on Management of Data*, pages 491–502.

[Doorenbos et al., 199] Doorenbos, R., Ezioti, O., and Weld, D. (199). A scalable comparison-shopping agent for the World Wide Web. In *Proceedings of the 1st Conference on Autonomous Agents*, pages 39–48.

[Fontes and Silva, 2004] Fontes, A. and Silva, F. (2004). SmartCrawl: a new strategy for the exploration of the hidden Web. In *Proceedings of the 6th Workshop on Web Information and Data Management*, pages 9–15.

[Golgher et al., 2000] Golgher, P., Laender, A., da Silva, A., and Ribeiro-Neto, B. (2000). AS-ByE: uma ferramenta baseada em exemplos para especificação de agentes para coleta de documentos web. In *Anais do XV Simpósio Brasileiro de Banco de Dados*, pages 217–231.

[Hedley et al., 2004] Hedley, Y., Younas, M., James, A., and Sanderson, M. (2004). A two-phase sampling technique for information extraction from hidden-web database. In *Proceedings of the 6th Workshop on Web Information and Data Management*, pages 1–8.

[Lage et al., 2004] Lage, J., da Silva, A., Golgher, P., and Laender, A. (2004). Automatic generation of agents for collecting hidden web pages for data extraction. *Data an Knowledge Engineering*, 49(2):177–196.

[Liu et al., 2004] Liu, H., Milios, E., and Janssen, J. (2004). Probabilistic models for focused web crawling. In *Proceedings of the 6th Workshop on Web Information and Data Management*, pages 16–22.

[Najork and Wiener, 2001] Najork, M. and Wiener, J. (2001). Breadth-first crawling yields high-quality pages. In *Proceedings of the 10th World Wide Web Conference*, pages 114–118.

[Qin et al., 2004] Qin, J., Zhou, Y., and Chau, M. (2004). Building domain-specific web collections for scientific digital libraries: a meta-search enhanced focused crawling method. In *Proceedings of the 4th Conference on Digital Libraries*, pages 135–141.

[Raghavan and Garcia-Molina, 2001] Raghavan, S. and Garcia-Molina, H. (2001). Crawling the hidden Web. In *Proceedings of the 27th Conference on Very Large Data Bases*, pages 129–138.

[Reis et al., 2002] Reis, D., Araújo, R., da Silva, A., and Ribeiro-Neto, B. (2002). A framework for generating attribute extractors for web data source. In *Proceedings of the 9th Symposium on String Processing and Information Retrieval*, pages 210–226.

[Reis et al., 2004] Reis, D., Golgher, P., da Silva, A., and Laender, A. (2004). Automatic web news extraction using tree edit distance. In *Proceedings of the 13th World Wide Web Conference*, pages 502–511.

[Selkow, 1977] Selkow, S. (1977). The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186.

[Turmo et al., 2006] Turmo, J., Ageno, A., and Català, N. (2006). Adaptive information extraction. *ACM Computing Surveys*, 38(2):#4.

[Yang, 1991] Yang, W. (1991). Identifying syntactic differences between two programs. *Software: Practice and Experience*, 21(7):739–755.

[Zhai and Liu, 2005] Zhai, Y. and Liu, B. (2005). Web data extraction based on partial tree alignment. In *Proceedings of the 14th World Wide Web Conference*, pages 76–85.

[Zhang et al., 2004] Zhang, Z., He, B., and Chang, K.-C. (2004). Understanding web query interfaces: Best-effort parsing with hidden syntax. In *Proceedings of the 30th Conference on Management of Data*, pages 107–118.