An Automated Algorithm for Extracting Website Skeleton

3 AUTHORS:

Zehua Liu
Zendesk
29 PUBLICATIONS **207** CITATIONS

SEE PROFILE

Wee Keong Ng
Nanyang Technological University
212 PUBLICATIONS **1,888** CITATIONS

SEE PROFILE

Ee-Peng Lim
Singapore Management University
394 PUBLICATIONS **4,942** CITATIONS

SEE PROFILE

An Automated Algorithm for Extracting Website Skeleton

Zehua Liu, Wee Keong Ng and Ee-Peng Lim
Centre for Advanced Information Systems
School of Computer Engineering
Nanyang Technological University
Singapore, 639798
{aszhliu, awkng, aseplim}@ntu.edu.sg

December 8, 2003

Abstract

The huge amount of information available on the Web has attracted many research efforts into developing wrappers that extract data from webpages. However, as most of the systems for generating wrappers focus on extracting data at page-level, data extraction at site-level remains a manual or semi-automatic process. In this paper, we study the problem of extracting website skeleton, i.e. extracting the underlying hyperlink structure that is used to organize the content pages in a given website. We propose an automated algorithm, called the SEW algorithm, to discover the skeleton of a website. Given a page, the algorithm examines hyperlinks in groups and identifies the navigation links that point to pages in the next level in the website structure. The entire skeleton is then constructed by recursively fetching pages pointed by the discovered links and analyzing these pages using the same process. Our experiments on real life websites show that the algorithm achieves a high recall with moderate precision.

1 Introduction

1.1 Motivation

There is a vast amount of information exists in public websites. This information is often presented in a manner that is easily accessible through manual browsing. To make the information also accessible to software programs, much research [1, 3, 8, 5, 14, 15] has been carried out to generate software systems, called *wrappers*, that automatically identify and extract the information that users are interested in from webpages and convert the extracted data into structured format.

Wrappers created by most existing wrapper systems only deal with one document, mainly because identifying relevant webpages is traditionally considered as the task of information retrieval (IR). However, information existing within a single page may not be complete by itself. Sometimes, a complete set of information can only be accessed by navigating several linked webpages. More importantly, pages within a website often form some coherent structure that reflects the implicit logical relationship between information exists in different pages. Therefore, it is tempting to extract the implicit structure hidden behind the interlinking between the webpages in a site and to prepare such structure together with

the contents of the pages for the wrapper generation task. For this purpose, wrappers have to be equipped with the ability to traverse links to extract information from multiple pages.

Manually creating wrappers is known to be difficult and knowledge intensive [8, 15]. Doing that for wrappers that can follow hyperlinks to extract information from different pages is even more difficult. Wrapper generation at page level can be done automatically without any human intervention [5, 1]. However, these automated systems do not address the problem of extracting from a set of linked webpages. Several *supervised wrapper generation* systems [3, 14] provide visual tools to help users interactively generate site-level wrappers, i.e. wrappers that have the capabilities to traversing hyperlinks. A typical problem with these systems is that they focus more on providing helps to specify or to derive extraction rules for data within a page. Less attention has been paid to the extraction of hyperlinks and when they do, the hyperlinks are often treated separately. As a result, the process of identifying and specifying the higher level structure of the information, which is often exhibited through the hyperlink structure, has to be done manually. This greatly reduces the degree of automation of the overall wrapper generation process.

To address the above problems, more automated ways of discovering the skeleton of the target website are needed. The *skeleton* here refers to the hyperlink structure that is used in the website to organize the *core contents* that the site is providing. For example, a newspaper website provides news articles as its core contents and organizes news into different sections and subsections in different pages. The hierarchical structure formed by the interlinking between the homepage and the pages containing different sections and subsections is considered as the skeleton of the site.

To analyze the hyperlink structure of a website, existing information retrieval and WWW searching methods, such as HITS [10] and PageRank [4], use eigenvector-based techniques to discover important pages from a set of candidate pages. These discovered pages are usually separated. An assembling process is required to organize these pages into a meaningful structure. However, this is non-trivial since some of the intermediate pages that connect these important pages may not be discovered. Thus these methods often fail to find the skeletons of websites.

In this paper, we propose the SEW[1] algorithm that automatically discovers the skeleton of a website. Instead of a bottom-up approach (finding out the content pages first and organize them later), the proposed algorithm starts from the homepage of a website and discovers the links to the pages in the next level in a top-down manner. It relies on a combination of several domain independent heuristics and features to identify the most important set of links within each page and organizes the pages into a hierarchy that captures the logical structure of the site.

---

[1]SEW stands for Skeleton Extraction from Websites

1.2 Applications

Such a website skeleton extraction algorithm would be a useful complement to both automated page-level wrapper generation systems [5, 1] and supervised wrapper generation systems [3, 14]. For the former, the pages appearing as leaf nodes (i.e. those pages containing the core contents) in the extracted skeleton can be supplied as input to produce page-level wrappers; thus making it possible to compose a site-level wrapper from a site skeleton and a set of page-level wrappers. For the latter, the extracted skeleton may serve as a starting point where users can refine the skeleton and specify extraction details of finer granularity in each individual page; hence greatly improving the degree of automation in wrapper generation.

Besides its direct applications in wrapper generation, the algorithm would also be useful to a number of Web mining applications. *Web classification* has mainly focused on individual webpages or a small set of pages. Ester et al. [7] proposed to classify based on *Web site tree*, which is simply the entire hyperlink structure of the website. Given the skeleton of a website, classification can now be performed at a larger range of granularity (from page-level to site-level) and on a more focused (and probably more important) sub-structure of the entire hyperlink structure. The discovered skeleton may also be used together with Web server logs for *Web usage mining* and personalization. *Mining structures of websites*, which has not been well studied previously, may take advantage of the skeletons extracted to perform structure mining. Another example is *website comparison* in terms of content structure using their skeletons, which may help to discover hidden website replication based on content structures instead of presentation styles.

Applications of the algorithm are not limited to wrapper generation and Web mining. In fact, we believe that any task related to website structure may take advantage of it. For example, it is possible to periodically extract the skeleton of a website and detect the changes in structure. This is important to the task known as *wrapper maintenance*, where detection of structural changes is required (and yet still remains unexplored). The extracted skeleton may also help Web crawlers to *prioritize crawling* by equipping them with knowledge of what pages are about the core contents of a particular website.

The remainder of the paper is organized as follows. Section 3 elaborates the SEW algorithm in details. The preliminary experimental results is presented in Section 4. Section 5 compares our work with related systems. Finally, Section 6 gives some concluding remarks.

2 Problem Definition

In this paper, a *website*, or simply a *site*, refers to a set of interlinked webpages that can be reached from some starting URL, such as "http://news.bbc.co.uk/" or "http://www.cnn.com/". The starting URL of a website is called the *base URL* of the site[2]. The page with the base URL is called the *homepage* of the

---

[2]In general, it is possible for a site to have more than one URLs that can serve the purpose of a base URL. In such cases, we take the most commonly used one as the base URL.

3

website. The *core contents* of a website refer to the information that the majority of users visiting the website are interested in. For example, news articles in a online newspaper website are the core contents of the site whereas the advertisements or the "about us" information are not.

Pages containing core contents are called *content pages* and pages containing links to content pages are called *navigation pages*. A page that contains links to a set of navigation pages is also considered as a navigation page. A page can be both a content page and a navigation page if it contains core contents as well as links to other content pages. For example, the webpage about "World" news in CNN.com is considered as a content page because it provides news articles on events happening around the world. The same page also contains a set of links to the subsections under the "World" section, such as "Europe" and "Asia Pacific". Thus, it is also a navigation page. Among all the links in a navigation page, those that point to content pages or other navigation pages are called *navigation links*, which are collectively called the *navigation link set* of the page.

The *skeleton* of a website refers to the hyperlink structure that content pages in a website are organized into. We assume that there is only one skeleton for a website and the skeleton is of a tree-like structure, where leaf nodes are content pages and internal nodes are navigation pages containing links to their child nodes that could be either other navigation pages (internal nodes) or content pages (leaf nodes). Since navigation pages may also contain core contents, internal nodes could be content pages, too. With all the definitions above, the website skeleton extraction problem is simply defined as: *given a website, find its skeleton*.

To distinguish the wrappers that we are trying to build from others', we call wrappers that extract information from a website *site-level wrappers* and those that extract data from only one (content) page *page-level wrappers*.

3 The Wese Algorithm

In this section, we present the SEW algorithm for the website skeleton extraction problem. Section 3.1 gives an overview of the algorithm by walking through an example. Section 3.2 to 3.4 then elaborate the algorithm in details.

3.1 Overview

Given the homepage of a website, the process of discovering navigation links from this page is divided into two steps: 1) finding *candidate link sets* and 2) identifying navigation links from the candidates. Candidate link sets are those sets of links in a page that could potentially be the navigation links. Consider the homepage of CNN.com[3] (shown in Figure 1), there are several clusters or groups of links where links within each group are structurally similar (in terms of DOM tree) to each other, as highlighted in blocks

---

[3]http://www.cnn.com

Group 4

Group 1
Home Page
World
U.S.
Weather
Business at cnnmoney
Sports at SI.com
Politics
Law
Technology
Science & Space
Health
Entertainment
Travel
Education
Special Reports

Link a
Link b

More bad weather feared for tornado survivors | Video
Peterson judge refuses to recuse himself | Video
GOP moralist Bennett: 'I have done too much gambling'
Jessica Lynch can't remember ambush | Video
Berkeley turns away students from SARS regions | Video
CNN/Money: Turner sells more than half of his AOL stock
Britain's Prince Edward, wife Sophie expecting baby
CNNRadio: Latest updates

Odd Links

Group 2
Video
Newswatch
E-Mail Services
CNN To Go

CNN.com

Did Saddam's son make a billion dollar withdrawal?

Group 6
DJIA
NAS
S&P

Video: ☀ Inside the life of Qusay Hussein
Report: Qusay used trucks to take $1 billion
Rumsfeld expected to cite mobile bio-lab in Iraq
Iraqi info minister makes musical debut

Group 3

Special Report: Life After War | Transition Of Power
Video: ☀ Iraq's top athletes beaten, tortured
Video: ☀ Sanctions may slow rebuilding efforts
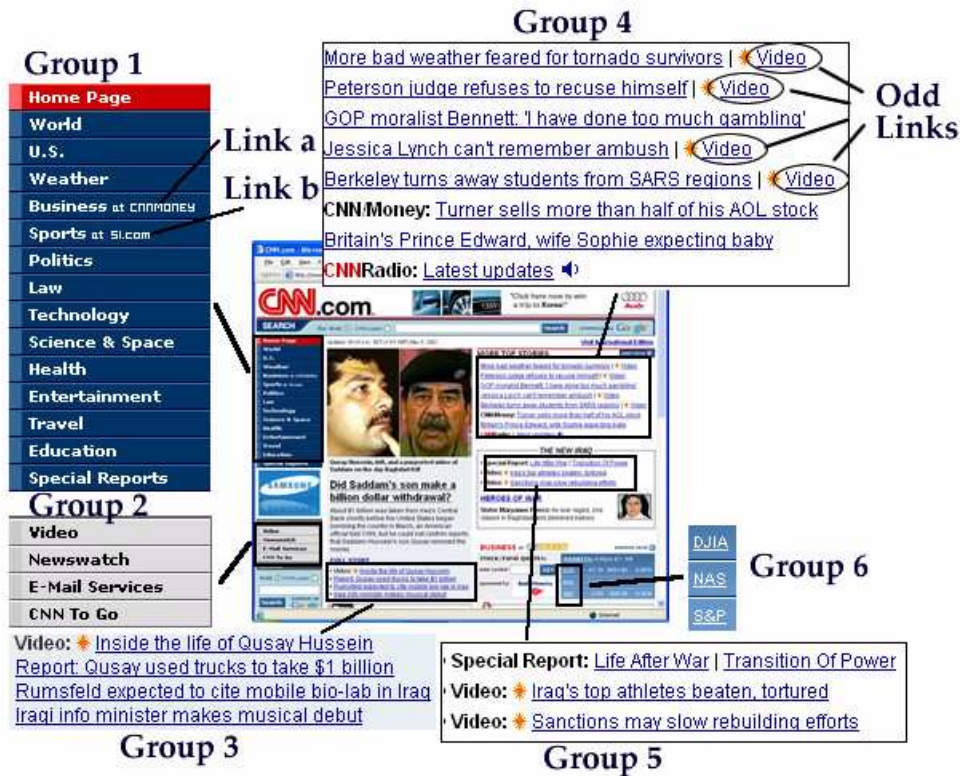
Group 5

Figure 1: The Homepage of CNN.com

in the figure. Each of these blocks may be considered as a candidate of the navigation link set of this page.

However, not all the link sets obtained by calculating structural similarity are good candidates. Several heuristics have been developed to prune the candidates based on observations of characteristics of navigation link sets. For example, group 6 in Figure 1 has only three links; thus is very unlikely to be a navigation link set, because a page usually contains more than three navigation links. In addition, within each group of links, there might be some links that are unlikely to be navigation links. For example, in Figure 1, link a and b in group 1 appear abnormally comparing with other links in the group. These two links should be removed to make the group a better candidate link set. The removal of these two links is also supported by the fact that they point to the same pages as two other links (Business and Sports) in the same group. After applying these heuristics, some link sets are discarded and some links within the remaining link sets are removed and a collection of candidate link sets is obtained. In the example in Figure 1, the candidate link sets are group 1, 2, 3, 4, and 5 (i.e. group 6 removed), with some odd links removed from group 1 and 4.

After obtaining these candidate link sets, the next step is to identify the link set that is the most likely to be navigation links. This is done by looking at several features of each candidate and checking whether

they have feature values similar to that of the navigation link sets that we have discovered. Examples of such features include average number of words in anchor texts and type of URL pointed by the links. For instance, in Figure 1, the average length and the number of words in the anchor texts of group 1 and 2 are quite close to the expected values of observed navigation link sets; while those of the rest of the groups are not. In addition, the URLs of the links in group 1 (not shown in the figure) are all relative URLs; whereas those of group 2 have some absolute URLs mixed together with relative ones. Based on navigation link sets found earlier (or training data set), the URLs of their links tend to have the same type of URL. Thus we are confident that group 1 is more likely to be the navigation link set than group 2 based on this feature. To decide which candidate is the best, the confidence scores obtained from all the features are combined to give a final confidence. In the above example, both features prefer group 1 to others. Therefore, group 1 is selected as the navigation link set of this page.

The above process discovers the navigation links in the homepage of CNN.com. As each navigation link points to either a navigation page or a content page, all pages pointed by the discovered navigation links are retrieved and the same process of discovering navigation link set is applied to each of these pages. For pages other than the homepage, more pruning of candidates can be performed by removing those link sets similar to the navigation link sets that have been discovered in other pages. Pages without any candidate link set or those with candidates that give confidence scores lower than some threshold are considered as content pages. For pages where navigation links are found, the relevant pages are again retrieved and analyzed in a similar manner. The overall process terminates when no more new pages are to be retrieved for analysis. The skeleton of the website is then constructed by treating the homepage as the root node and the pages directly retrieved from the homepage as nodes in the second level and navigation links in the homepage as edges to these nodes. Subtree rooted from the nodes in the second level are constructed recursively. For the case of CNN.com, we obtain a tree with maximum depth of 3, where the second level contains nodes such as "World", "U.S.", "Weather", etc, and the node "World" has children "Asia", "Africa", "Europe", etc.

3.2   Finding Candidate Link Sets

As it is generally very difficult to directly identify the set of navigation links from all links in a page, we divide this process into two steps. First, we group the links into sets and select those that are more likely to be navigation link set for further investigation. This is the step for *finding candidate link sets*. Next, the *navigation link set identification* step examines the candidate sets and chooses the best one as the navigation link set. This subsection describes the first step in details; the second step is discussed in next subsection (Section 3.3).

In the local context of a few pages (one page in the case of the homepage), a single link by itself provides little information. It is difficult to tell whether a link is a navigation link by looking at the link alone. Therefore, while other link-based analysis methods (e.g. HITS and PageRank) analyze links one

by one, we choose to analyze links in groups. We consider a set of links as a candidate and explore the common characteristics exhibited by links within each set and their relationships to further prune them. The same strategy of considering links in groups is also used in the next step to pick out the best candidate.

Two issues are involved in finding a good collection of candidate link sets out of all links in a page. Firstly, we have to decide how to cluster the links into groups. Secondly, since the number of groups and the number of links in each group might be large, we have to prune the groups. Before going into the details of the algorithm that addresses these two issues, we briefly describe the XHTML data model for representing web documents.

3.2.1   The XHTML DOM Tree Model

Given an HTML webpage, we first convert it into XHTML format (by cleaning bad and ill-formed tags and inserting end tags for bachelor tags[4]) and parse the XHTML page into a DOM tree by treating it as an XML document. Each start tag in the XHTML document (e.g. <TABLE> and <A>) is represented as a node. The label of the node, called NodeName, is the same as the name of the start tag (e.g. TABLE and A). Attributes of a start tag are converted to attributes of the corresponding node. Tags appearing between the occurrences of a start tag and the corresponding end tag are treated as child nodes of the node representing the start tag. Texts are converted to text nodes.

In this DOM tree model, links (those tags with tag name A) are represented as nodes with NodeName A, which we call *link nodes*. Link nodes are numbered in the order that they are visited in a pre-order, depth-first, left-to-right traversal of the DOM tree. It is the same as the order in which the A tags appear in the HTML source document. Among the attributes of a link node, the one named HREF, alternatively called the URL attribute, is of special interest because it is the URL of the page that the link is pointing to. Each link node is also associated with an additional attribute SecName that represents a short description of the page that the link is pointing to. The value of SecName is obtained by concatenating all the anchor texts of the link. If the anchor is an image, the ALT attribute of the IMG tag is taken as the value. If the value of the ALT attribute is not set, the value of SecName is an empty string.

3.2.2   Generating Candidate Link Sets

Figure 2 depicts the algorithm used to generate candidate link sets from a page. After obtaining the XHTML data model DOMTree, all the link nodes are identified and placed into a list, denoted as Al-lLinks, in the order of their numbering. An iterative process (the while loop in line 6–22) is then started to pull link nodes out of AllLinks and generate candidate link sets until AllLinks is empty.

---

[4]http://tidy.sourceforge.net/

```
 1: function GENCANLINKSET(URL, CurNode)
 2:    SRC = CLEANANDREPAIRHTML(URL)
 3:    DOMTree = PARSEXHTML(SRC)
 4:    CanLinkSets = {}
 5:    AllLinks = FINDALLLINKNODES(DOMTree)
 6:    while AllLinks is not empty do
 7:       RefLink = first link node in AllLinks
 8:       CanLinks = FINDLINKNODESATSAMELEVEL(RefLink, AllLinks)
 9:       if CanLinks.size() > 0 then
10:          add RefLink into CanLinks
11:          PathSet = GROUPBYPATH(CanLinks)
12:          for each linkset LinkSet in PathSet do
13:             PRUNELINKSET(LinkSet, CurNode)
14:             if MinNL < LinkSet.size() < MaxNL then
15:                add LinkSet into CanLinkSet
16:             end if
17:             remove links in LinkSet from AllLinks
18:          end for
19:       else
20:          remove RefLink from AllLinks
21:       end if
22:    end while
23:    return CanLinkSets
24: end function
25:
26: function PRUNELINKSET(LinkSet, CurNode)
27:    PHANCHOR(LinkSet)
28:    PHLINKDUP(LinkSet)
29:    PHSTYLE(LinkSet)
30:    PHDISDIFF(LinkSet)
31:    PHMAXNUMWORD(LinkSet)
32:    PHLINKSETDUP(LinkSet,CurNode)
33: end function
```

Figure 2: The GENCANLINKSET Function

Clustering Links At each iteration, the first link node in AllLinks, denoted as RefLink, is taken out and all link nodes remaining in AllLinks that have the same depth (i.e. number of nodes in the path to the root node in the XHTML DOM tree) as RefLink are found and added into the set CanLinks. An initial clustering of link nodes in CanLinks is performed by calling the function GROUPBYPATH, which simply divides the nodes into clusters where nodes in each cluster have the same path to the root node in the XHTML DOM tree.

Clustering links by path to produce potential candidates is based on two observations. Firstly, link nodes corresponding to navigation links within a page are almost always located at the same level in the DOM tree. Secondly, since navigation links usually are presented in menu-like styles, they often have the same path in the DOM tree. It should be noted that physical proximity of links is not used, because sometimes navigation links may be regularly distant from each other. For example, a two-level menu may sometimes be used, where the links at the first level are separated by some links at the second level. In such a case, physical proximity may easily put links from two levels into one group; whereas the clustering by path approach can effectively distinguish the first level links from those at the second level, as they are very likely to have different paths or even appear at the different levels in the DOM tree.

Pruning Candidates As mentioned earlier, the initial candidate link sets produced may contain too many groups or groups too large in size. Therefore, at each iteration, after obtaining PathSet, which is a set of initial candidate link sets, we apply several simple heuristics to prune each initial candidate LinkSet (by calling the function PRUNELINKSET) and only consider the candidate if it survives after the pruning. The *pruning heuristics* used here all correspond to some observations of characteristics that navigation link sets have. Currently, the pruning heuristics that have been implemented include:

- **Anchor** The anchor of a link node refers to its child nodes. Most navigation links in a page are formatted similarly; thus having the same type of presentation style in their anchor contents. Based on this observation, the Anchor heuristic clusters the link nodes in LinkSet into groups by the type of their anchor contents and only keeps the group with the largest number of links. Anchor contents are considered as different types if their presentation styles are different. Since the most commonly used styles are plain texts and images (IMG tag), we only consider these two types here.

- **LinkDup** Link nodes in the same LinkSet with the same SecName or URL are considered redundant and only one will be kept. A pairwise comparison of the SecName and URL attributes of link nodes within the group is performed. If duplication of links is found, the one that appears later will be discarded.

- **Style** Navigation links usually not only have the same path but also same Cascading Style Sheet (CSS) style and other presentation-related attributes in the nodes along the path to the root node.

The Style heuristic further clusters the link nodes in LinkSet by comparing the CSS style of the nodes in the path. Other presentation related attributes that are used include SIZE and COLOR of the FONT tag, BGCOLOR of the TR and TD tags, etc. Only the largest cluster will be kept.

- **DisDiff** Although physical proximity is not appropriate for generating the initial candidates, it is still useful for pruning the links. The DisDiff heuristic computes the distances between every two consecutive link nodes (in their numbered order). If some distance is found to be statistically much larger than the rest, it is considered as an indication that the LinkSet might be a combination of two or more candidate link sets. Therefore, if there is a statistically larger distance, the LinkSet is divided into two sets by the position of that distance in the link node sequence and only the first subset is retained.

- **MaxNumWord** We observe that navigation links usually contain few number of words in their anchor text (typically one or two words). Thus after all the above heuristics have been applied, the average number of words in the SecName attribute of all remaining link nodes in the group is calculated. If this number is greater than or equal to a predefined threshold (4 in our experiments), the entire group of link nodes is discarded. This heuristic is applied after those introduced above because if applied earlier, the "bad" link nodes that are supposed to be removed by previous heuristics may introduce noise (longer SecName) that leads to the removal of a valid candidate link set.

- **LinkSetDup** If the page that is currently being analyzed is not the homepage, we have to ensure that the navigation link sets generated from previously analyzed pages are not reconsidered here. This heuristic simply checks whether LinkSet is the same as some navigation link set discovered earlier (from other pages). If yes, it will be discarded.

The heuristics discussed above, when applied to a potential candidate link set, either remove some "bad" candidate link from the set or discard the entire set. As an additional heuristic, we only consider candidate link set that has the size that is not too small and not too large (line 14 in Figure 2). LinkSet with the size in the range between MinNL and MaxNL is added into the candidate list CanLinkSets. All link nodes in LinkSet are removed from AllLinks. The same candidate pruning process is then repeated on other link sets resulted from the GROUPBYPATH function call in line 11 in Figure 2. When all initial candidate link sets have been pruned, another iteration of generating initial candidates and pruning them is started. The entire iteration process (line 6–22) terminates when AllLinks becomes empty. At this point, the function returns CanLinkSets, which contains all the generated candidate link sets.

3.3 Identifying Navigation Link Set

With the candidate link sets generated from the previous step, the next task is to identify the navigation link set from all the candidates. This requries some sort of ranking mechanism such that all the candidates

can be ordered in terms of the likelihood of each one being the navigation link set. The candidate ranked at the top can then be selected.

We adopt a feature-based approach to evaluate the candidates based on a set of predefined features. We compute the value of each feature of each candidate link set and compare the value with the norm value of that feature. The closer the feature value to its norm, the more confident we are that the candidate is the right navigation link set. Finally, the confidence scores obtained from all features are combined to give the final confidence score of a candidate link set being the navigation link set. Using this combined confidence score, all candidates can be ranked.

3.3.1 Link Set Features

Similar to the pruning heuristics, the features are derived based on observations on the common characteristics of navigation links. Given a link set, the value of a feature is a numeric value. For each feature, we compute, based on observed navigation link sets, the average feature value and the standard deviation of the value. This average and standard deviation pair is used to calculate the likelihood, also called the *confidence score*, of a particular link set being a navigation link set, given the feature value that it has. The calculation of likelihood is based on the normal distribution probability density function, as shown in the function NORMDIST in Figure 3 (line 17–20). Note that this is a standardized normal distribution so that the effect of the size of the standard deviation value can be eliminated and the computed confidence scores for different features can be directly compared and combined. Based on the computed confidence score, the closer a feature value is to the average, the higher the score is; thus the more likely that a link set having that feature value is a navigation link set.

Currently, we have defined six features: NumWordName, LenName, VarLenName, URLType, VarURLNumDir, and URLContain. The first three make use of the characteristics of the anchor texts of links. Given a link set, the value of the NumWordName feature is the average number of words in the SecName attribute of all link nodes in the set. Similarly, LenName calculates the average length (i.e. number of characters) of the SecName attribute. Rather than taking simple average, the VarLenName feature computes the variance of the length of SecName among all link nodes.

The last three features are based on the observation that the URL attributes (HREF) of navigation links tend to exhibit similar patterns. To measure the similarity of values of the URL attributes among all link nodes, the URLType feature compares the types of the URLs. Five types of URLs have been defined: *relative* (the link is relative to the current directory), *absolute* (the link is relative to the current site), *cross-site* (the link is independent of the current site), *anchor* (links with '#'), and *others* (all other types of link, e.g. those involving JavaScript functions). We find out the type with the largest number of link nodes. We then count the number of link nodes of this type and normalize the count with the total number of nodes in the link set.

The VarURLNumDir feature is derived based on the fact that the number of levels of directories in

the URL attributes of navigation link sets tend to be the same. To quantize this tendency towards the same value, we compute the variance of the number of levels of directories of all links in a link set. For relative and anchor URLs, the URL is prefixed with the sub-path from the directory of the homepage to the current directory. For absolute and cross-site URLs, the URL without the HTML filename is considered. For URLs whose type is *others*, the number of levels is taken to be a significantly large number (1000 in our experiment).

The last feature URLContain counts, for each candidate, the number of links whose URLs are contained under the directory of the current page. This corresponds to the observation that the hierarchy in a skeleton is often related to the directory hierarchy, where the navigation links contained within a navigation page often exist within the sub-directories of the directory of that page. For example, the Entertainment page of CNN.com is in the directory /SHOWBIZ/ and the navigation pages from this page such as Movies and Music all exists in the subdirectories of /SHOWBIZ/ (/SHOWBIZ/Movies/ and /SHOWBIZ/Music/). The count is normalized by the number of links in each candidate set.

All these features are based on either anchor texts or the URL attributes of link nodes. The DOM tree structure is not utilized, mainly because it has already been heavily used to prune link nodes in candidates. Since the link nodes in the candidates all go through the pruning step, they should have similar characteristics in terms of DOM tree structure. Therefore, the DOM structure is unlikely to give much information in this step.

3.3.2 Combined Ranking

The confidence score of each feature is calculated independently, as they each have their own pre-computed average and standard deviation. Using the confidence scores of each feature, the candidate link sets can be ranked. However, the scores from different features might not agree on which candidate is the best, i.e. not all of them rank the same candidate link set as the top one. In fact, each feature may work well only for certain websites (or pages) but not others. Therefore, the ranking from different features have to be combined to provide a more accurate estimation of the navigation link set.

To combine the five heuristics, we consider an event $e_{i,j}$ represent the fact that the value of feature f_j of the ith candidate is consistent with the expected value of that feature. The confidence score $c_{i,j}$ of feature f_j of the ith candidate is the probability $P(e_{i,j})$ of event $e_{i,j}$ occurring. Thus, the problem of computing the combined confidence score c_i of the ith candidate is reduced to that of computing the combined probability $P(\wedge_j e_{i,j})$.

However, to compute $P(\wedge_j e_{i,j})$ exactly, a set of conditional probabilities is needed, which requires a large number of training data. For simplicity, we follow common practice by assuming that the observations of events for each candidate are independent[5]. The combined probability is thus calculated by

---

[5]Other dependency assumptions are also possible. See [11] for relevant discussion.

```
 1: function SELECTNAVLINKSET(CanLinkSets)
 2:     for each linkset $s_i \in$ CanLinkSets do
 3:        for each feature $f_j \in F$ do
 4:            $c_{i,j} = $ NORMDIST($f_j(s_i)$, $\mu_{F_j}$, $\sigma_{F_j}$)
 5:        end for
 6:        $c_i = $ COMBINECONF($\{c_{i,j} | 1 \leq j \leq |F|\}$)
 7:     end for
 8:     $c_k = $ MAX($\{c_i | 1 \leq i \leq |CanLinkSets|\}$)
 9:     $p = $ PROB($c_k$, $\mu_c$, $\sigma_c$)
10:     if $p \geq \tau$ then
11:        return $s_k$
12:     else
13:        return $\{\}$
14:     end if
15: end function
16:
17: function NORMDIST($f$, $\mu$, $\sigma$)
18:     $f_{st} = \frac{f-\mu}{\sigma}$
19:     return $\frac{1}{\sqrt{2\pi}} e^{-\frac{f_{st}^2}{2}}$
20: end function
21: function PROB($f$, $\mu$, $\sigma$)
22:     return $\int_{-\infty}^{f}$ NORMDIST($f'$, $\mu$, $\sigma$)$df'$
23: end function
24: function COMBINECONF($C$)
25:     return $\prod_{c_i \in C} c_i$
26: end function
```

Figure 3: The SELECTNAVLINKSET Function

$$P(\wedge_j e_{i,j}) = \prod_{j=1}^{n} P(e_{i,j})$$

Therefore, the combined confidence is given by

$$c_i = P(\wedge_j e_{i,j}) = \prod_{j=1}^{n} P(e_{i,j}) = \prod_{j=1}^{n} c_{i,j}$$

Figure 3 depicts the algorithm for selecting the best candidate as the navigation link set. For each candidate link set, the algorithm calculates the confidence score of each feature and combines the confidence scores based on the independence assumption by multiplying the confidence scores from individual features (line 2–7, 26–28). After the combined confidence scores of all candidates have been calculated, the candidate with the highest score is picked out (line 8).

Given the best candidate, we can't just simply accept it as the navigation link set because we do not know whether the page that we are dealing with is a navigation page or content page. To determine this, the algorithm tests the score of the best candidate against the average μ_c and standard deviation σ_c

of combined confidence scores pre-computed from known navigation link sets. The test is performed by calculating the cumulative probability density p of the combined confidence score over a normal distribution curve determined by μ_c and σ_c (line 9–10, 22–24) and comparing p with a pre-defined threshold τ. If p is greater than τ, the candidate link set is accepted as the navigation link set and is returned by the function. Otherwise, the function returns an empty set.

3.4 The Algorithm

The two functions GENCANLINKSETS and SELECTNAVLINKSET together provide a way to determine navigation links of a given page. With this, we are ready to introduce the complete SEW algorithm that extracts the skeleton of a website.

The complete algorithm is shown in Figure 4. To start the skeleton extraction process, we invoke the SEWALGORITHM function passing the base URL of the target website as parameter. This functioon simply calls the FINDSKELETON function passing in the base URL. FINDSKELETON is a recursive function that, given a page, invokes GENERATECANLINKSETS and SELECTNAVLINKSET to discover the navigation links and, for each discovered navigation link, recursively calls itself with the link as parameter to discover more navigation links. The order in which nodes in the skeleton are generated is that of a pre-order traversal of the skeleton. At the end, SEWALGORITHM returns the root node of the skeleton.

The recursive call in FINDSKELETON returns when the function SELECTNAVLINKSET returns an empty set, i.e., the computed cumulative probability density p of the best candidate is less than the threshold τ. Therefore, the testing of the values of p against τ can be considered as the stopping criteria of the entire algorithm.

4 Experiments

The SEW algorithm has been implemented in C++ using Microsoft Visual C++. In this section, we describe the preliminary experiments that we have conducted on some real-life newspaper websites.

4.1 Dataset

In the experiments, the dataset consists of five websites from online newspaper domain. Table 1 lists the five websites together with the related statistics. All five websites provide online news that cover a diverse range of topics. The layout styles and locations of navigation links in the webpages in these websites are rather versatile, ranging from nicely formatted vertical menu on the left handside to links arranged in a grid-like table in the middle of a page. The websites were crawled and all experiments are performed on the crawled copy of the websites. The SecName and URL attributes of each navigation link are extracted and stored for computing the average and standard deviation of each feature.

```
 1: function SEWALGORITHM(BaseURL)
 2:    return FINDSKELETON(BaseURL)
 3: end function
 4:
 5: function FINDSKELETON(URL)
 6:    CurNode = empty node
 7:    CanLinkSets = GENCANLINKSETS(URL,CurNode)
 8:    if CanLinkSets is not empty then
 9:       NavLinkSet = SELECTNAVLINKSET(CanLinkSets)
10:       for each link NavLink in NavLinkSet do
11:          ChildURL = HREF attribute of NavLink
12:          ChildNode = FINDSKELETON(ChildURL)
13:          add ChildNode as a child of CurNode
14:       end for
15:    end if
16:    return CurNode
17: end function
```

Figure 4: The SEW Algorithm

Table 1: Dataset and Related Statistics

name	base URL	total no. of nav pages	max depth	no. of nav pages at L1	no. of nav pages at L2
Washington Post	www.washingtonpost.com	327	3	19 / 5.8%	162 / 49.5%
CNN News	www.cnn.com	119	3	15 / 12.6%	50 / 42.0%
New York Post	www.nypost.com	90	3	8 / 8.9%	36 / 40%
Washington Times	www.washtimes.com	50	2	18 / 36%	50 / 100%
BBC News	news.bbc.co.uk	24	3	13 / 54.2%	19 / 79.2%

The websites were manually inspected to figure out the skeleton. As shown in Table 1, the numbers of nodes in the skeleton (including navigation pages and content pages) vary among sites depending on the complexity of the website skeleton and the coverage of the news contents. The percentage of nodes within a maximum depth of 1 and 2 are also calculated.

4.2 Methodology and Performance Metrics

The SEW algorithm requires a set of parameters, including the threshold for accepting a candidate and the average and standard deviation of all features and the combined confidence score. These parameters have to be computed during a training stage. To test the performance of the algorithm on the dataset, we performed a *leave-one-out* (LOO) cross validation by taking out one website for testing and using the other four as training data. The results were averaged and the standard deviation calculated. To test the maximum performance of the algorithm, we also performed a *test-on-training-data* (TOT) by testing each website using parameters trained from all five websites. This should in theory yield a better result

than LOO since the training data will inevitably over-fit the testing data. For both LOO and TOT, the algorithm was run three times by limiting the maximum number of levels to be explored to 1, 2, and 3. These runs are denoted as LOO1, LOO2, LOO3, TOT1, TOT2, and TOT3 respectively. The threshold for accepting the best candidate link set was set to the lowest combined confidence score computed from the training data.

To measure the overall performance, for each website, the precision p_{oa} and recall r_{oa} of finding navigation links was computed as

$$p_{oa} = \frac{\textit{no. of correct nav page}}{\textit{no. of pages found}}$$

$$r_{oa} = \frac{\textit{no. of correct nav page}}{\textit{no. of pages in the site}}$$

and for each p and r pair, the F1-measure f is

$$f = \frac{2pr}{p+r}$$

In order to better understand the performance of the algorithm, we also computed two additional sets of precision and recall for the two sub-steps (*finding candidate link sets* and *identifying the navigation link set*). To test the step of finding candidate link sets, for each correctly identified navigation page p_i, let $lsmax_i$ be the candidate link set that contains the largest number of correct navigation links, we computed precision p_{cg} and recall r_{cg} of candidate link set selection by

$$p_{cg} = \frac{1}{N} \sum_{i=1}^{N} \frac{\textit{no. of correct nav links in } lsmax_i}{\textit{no. of links in } lsmax_i}$$

$$r_{cg} = \frac{1}{N} \sum_{i=1}^{N} \frac{\textit{no. of correct nav links in } lsmax_i}{\textit{no. of links in nav link set of } p_i}$$

where N is the number of correctly identified navigation pages that have child nodes in the skeleton of the website.

To test the step of identifying the navigation link set, for each website, we computed precision p_{ns} of navigation link set selection by

$$p_{ns} = \frac{\textit{no. of pages with correct nav link set picked}}{\begin{array}{c}\textit{no. of pages where some cand set}\\\textit{contains the correct nav link set}\end{array}}$$

4.3 Results and Discussion

Table 2 shows the overall performance of the algorithm by summarizing the results from all five websites. Each entry in the table is of the form "*average ± standard deviation*". For extraction of first level of the skeleton (LOO1 and TOT1), the results were rather encouraging – all navigation links had been correctly identified and no false positive produced. This indicates that the algorithm would perform better for sites

Table 2: Overall Performance

Method	p_{oa}	r_{oa}	f_{oa}
LOO1	1 ± 0.0	1 ± 0.0	1 ± 0.0
LOO2	0.663 ± 0.169	0.908 ± 0.118	0.752 ± 0.132
LOO3	0.446 ± 0.216	0.879 ± 0.205	0.566 ± 0.211
TOT1	1 ± 0.0	1 ± 0.0	1 ± 0.0
TOT2	0.658 ± 0.162	0.908 ± 0.118	0.749 ± 0.128
TOT3	0.415 ± 0.234	0.861 ± 0.250	0.536 ± 0.237

Table 3: Overall Performance by Site

	LOO1			LOO2			LOO3		
	p_{oa}	r_{oa}	f_{oa}	p_{oa}	r_{oa}	f_{oa}	p_{oa}	r_{oa}	f_{oa}
Washington Post	1	1	1	0.839	0.977	0.903	0.368	0.869	0.516
CNN News	1	1	1	0.621	0.745	0.678	0.396	0.526	0.452
New York Post	1	1	1	0.714	1	0.833	0.488	1	0.656
Washington Times	1	1	1	0.745	0.820	0.781	0.781	1	0.877
BBC News	1	1	1	0.396	1	0.567	0.195	1	0.327

with small skeleton. This observation is further supported by the 100% recall in Table 3 (showing the detail LOO results of each site) for Washington Times, New York Post and BBC News, which are the sites with smallest skeleton among all sites. Being able to perform well in extracting small skeleton is desirable because most of the websites will unlikely to have a structure as complex as that of Washington Post and CNN News.

It can also be seen that for the extraction of complete skeletons (LOO3 and TOT3), a recall of about 88% had been achieved. However, precision remaind relatively much lower, mainly due to the fact that for many leaf nodes in the skeleton, the algorithm failed to reject the candidates, which resulted in some candidates being considered as candidate link sets and generating subtrees of false positive rooted from those candidates. This implies that the stopping criteria used in the algorithm does not perform very well.

Precision was improved when the number of levels was limited to 2 (LOO2 and TOT2), because the number of leaf nodes was less; thus giving less chances for false positives in leaf nodes to be accepted.

Another observation is that TOT did not improve LOO; on the contrary, it performed slightly worse than LOO. This shows that the feature set that we derive is site-independent since the sites did not really benefited from the parameters trained using the sites themselves and the parameters obtained from training were not really affected by the source of the training data.

We believe that, in practice, typically useful in a supervised wrapper generation environment, a high recall value is more important because it is much more difficult to find out the correct navigation pages (in the case of low recall) than to remove the incorrectly identified navigation pages from a list (in the case of high recall but low precision). Meanwhile, since false positives often exist as a whole subtree, the removal of an incorrect internal node would eliminate a large number of incorrect nodes (those in the subtree). In addition, when used together with automated page-level wrapper generation systems, false

Table 4: Candidate Generation Performance

Method	p_{cg}	r_{cg}	f_{cg}
LOO1	1 ± 0.0	1 ± 0.0	1 ± 0.0
LOO2	0.994 ± 0.027	0.981 ± 0.078	0.986 ± 0.054
LOO3	0.981 ± 0.123	0.974 ± 0.134	0.976 ± 0.127

Table 5: Nav Link Set Selection Performance

Method	p_{ns}
LOO1	1 ± 0.0
LOO2	0.941 ± 0.096
LOO3	0.924 ± 0.106

positive could be easily detected because these systems will simply fail or return nothing indicating that the input pages are not the correct content pages. Therefore, we consider the results obtained in Tables 2 and 3 (high recall but moderate precision) quite satisfactory.

Table 4 depicts the experimental results on the performance of the algorithm in terms of candidate generation. The results were quite encouraging, as we achieved precision and recall of above 97%. In our experiments, almost all candidates generated contained the correct navigation link set, with only 1 or 2 exceptions for each website. The implication of such a good performance in candidate generation is that it could be used in an interactive environment to present the top-k ranked candidates (not just the best one) to allow the users to choose from; thus increasing the recall of the overall process to 100%.

The precision of navigation link set identification is shown in Table 5. This result showed that about 92% of the time the algorithm was able to pick out the correct navigation link set from the candidates. Since the precision and recall of the candidate generation step is above 97%, the precision of navigation link set identification is the major factor that impact the overall performance of the algorithm, especially the recall value. Nevertheless, we note that the algorithm worked extremely well when extracting only the first level of the skeleton.

5   Related Work

The development of the SEW algorithm has been mainly motivated by works in Wrapper Generation [1, 3, 5, 6, 13, 14], where toolkits have been built to automatically or semi-automatically generate software programs that are capable of automatically extracting data from webpages. As mentioned in Section 1, most existing wrapper generation systems only deal with information contained within a single page, although some of them [1, 5] can produce wrappers in a fully automated manner. Other systems [2, 3] allow construction of models that represent information at site-level. But the degree of automation in these systems is relatively low (manual or semi-automatic). In comparison with these systems, the SEW algorithm distinguishes itself by providing a fully automated way to discover the hidden structure (skeleton) of a website. Automated algorithms such as those in [1, 5] are nice complements to the SEW

algorithm since they work at page-level while SEW works at site-level by treating pages as basic units. The combination of both would provide a completely automated solution to wrapper generation with a broader coverage of granularity, from data distributed across an entire website to those inside a single page.

Two other systems addressed similar problems to that of the website skeleton extraction. LAMIS [9] is an entropy-based link analysis algorithm that tries to discover *informative structure* of news websites. The informative structure that it discovers consists of a set of TOC (table of contents) pages and a set of article pages. These two types of pages, although limited to news domain, correspond roughly to navigation pages and content pages defined in Section 2. LAMIS computes the entropy of each link and use it as weight in the HITS algorithm [10] to analyze all pages to find out the most probable TOC pages. Comparing with the top-down approach of finding navigation pages in SEW, this HITS-based algorithm tends to generate more false positives and false negatives since it looks at all pages in a site while SEW only deals with links in the pages that it analyses. In addition, it is not clear how all the discovered TOC pages can be organized to form an informative structure of a website.

Li et al. [12] proposed a system for constructing multi-granular and topic-focused website maps. The approach is based on discovery of entry pages to *logical domains* (a set of self-contained pages) within a physical domain (a website). Multi-granular and topic-focused site maps can be constructed by adjusting the weights that determine the importance of an entry page and displaying only the more important pages in the map. The organization of entry pages into site map structure is based on the physical directory structure, which is only one of the six features that were used in our algorithm (the URLContain feature). One disadvantage is that some intermediate navigation pages may be discarded if they are not ranked important; thus resulting in a large number of disconnected subtree of pages that could only be blindly clustered and organized under a single root node. Nevertheless, the proposed weight assignment techniques may also be used in SEW to produce a customized skeleton tailored to certain topic or certain required granularity of details.

WWW Search algorithms, such as PageRank [4] and HITS [10], rely on the link structure of webpages to assign importance scores to the pages. These scores can be used to answer user queries by selecting the most relevant pages based on both the query keywords and importance scores. The importance scores in these systems measure mostly the popularity or authority of pages, which roughly corresponds to the likelihood that a page is a (good) content page. Thus only some content pages can be accurately found and no navigation pages are found. Although HITS also introduces the concept of a *hub* that captures part of the meaning of a navigation page, the importance score of a hub is closely related to how many *authorities* that it links to. As a result, navigation pages pointing to content pages that are unpopular will not be discovered. One important characteristic that distinguishes the SEW algorithm from these algorithms is that it analyzes links in groups, which gives it much more information and allows it to extract more interesting features of the links. This larger amount of information enables SEW to

performs well in a local context (e.g. a small website) as opposed to a global context (e.g. searching the entire Web), where link analysis based algorithms tend to perform better. In addition, the pages returned by these link-based algorithms are unorganized. Extra efforts are still required to produce a meaningful structure out of the list of search results, which sometimes may not be possible.

6   Conclusion

In this paper, we study the website skeleton extraction problem, which is an important step in wrapper generation for Web information extraction. We present the SEW algorithm for automatically discovering the skeleton of a website. The algorithm works in a recursive manner by applying a two-step process to discover the navigation links in a page and retrieving pages from the links to discover more navigation links. The two-step process involves generating candidate link sets and selecting the best candidate as the navigation link set. All the navigation links discovered and the pages retrieved form the skeleton of the website. Our preliminary experiments on real life newspaper websites showed that the algorithm performs well in recalling most of the navigation pages. However, it has yet to achieve a good precision.

As part of the future work, we would like to improve the algorithm to achieve better precision. In particular, techniques are required to prevent the algorithm from retrieving too many incorrect pages. One possible way is to have better features that distinguish navigation links from others so that incorrect link sets would be more likely to be rejected.

In addition, more extensive experiments have to be conducted to fully test the performance of the algorithm and to provide more hints on how it can be improved. This would include testing on larger datasets and on websites of categories other than newspaper websites.

References

[1] Arvind Arasu and Hector Garcia-Molina. Extracting Structured Data from Web Pages. In *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD 2003)*, San Diego, California, USA, June 9–12 2003. ACM Press.

[2] Paolo Atzeni, Giansalvatore Mecca, and Paolo Merialdo. To Weave the Web. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB 97)*, pages 206–215, Athens, Greece, August 25–29 1997.

[3] Robert Baumgartner, Sergio Flesca, and Georg Gottlob. Visual Web Information Extraction with Lixto. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 119–128, Roma, Italy, September 11–14 2001.

[4] Sergey Brin and Lawrence Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW 7)*, pages 107–117, Brisbane, Australia, April 14–18 1998.

[5] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. RoadRunner: Towards Automatic Data Extraction from Large Web Sites. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001)*, pages 109–118, Roma, Italy, September 11–14 2001.

[6] David W. Embley, Douglas M. Campbell, Yuan Stephen Jiang, Stephen W. Liddle, Deryle W. Lonsdale, Yiu-Kai Ng, and Randy D. Smith. Conceptual-model-based data extraction from multiple-record Web pages. *Data & Knowledge Engineering*, 31(3):227–251, 1999.

[7] Martin Ester, Hans-Peter Kriegel, and Matthias Schubert. Web Site Mining: A new way to spot Competitors, Customers and Suppliers in the World Wide Web. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD 2002)*, pages 249–258, Edmonton, Alberta, Canada, July 23–26 2002. ACM.

[8] Joachim Hammer, Héctor García-Molina, Junghoo Cho, Arturo Crespo, and Rohan Aranha. Extracting Semistructured Information from the Web. In *Proceedings of the Workshop on Management of Semistructured Data*, pages 18–25, Tucson, Arizona, USA, May 16 1997.

[9] Hung-Yu Kao, Shian-Hua Lin, Jan-Ming Ho, and Ming-Syan Chen. Entropy-based link analysis for mining web informative structures. In *Proceedings of the 2002 ACM CIKM International Conference on Information and Knowledge Management (CIKM 2002)*, pages 574–581, McLean, VA, USA, November 4–9 2002. ACM.

[10] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1999.

[11] Nicholas Kushmerick. Wrapper verification. *World Wide Web Journal*, 3(2):79–94, 2000.

[12] Wen-Syan Li, Necip Fazil Ayan, Okan Kolak, Quoc Vu, Hajime Takano, and Hisashi Shimamura. Constructing Multi-Granular and Topic-Focused Web Site Maps. In *Proceedings of the Tenth International World Wide Web Conference (WWW 10)*, pages 343–354, Hong Kong, China, May 1–5 2001. ACM.

[13] Ling Liu, Calton Pu, and Wei Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *Proceedings of the 16th International Conference on Data Engineering (ICDE 2000)*, pages 611–621, San Diego, California, USA, February 28 – March 3 2000.

[14] Zehua Liu, Feifei Li, Wee Keong Ng, and Ee-Peng Lim. A Visual Tool for Building Logical Data Models of Websites. In *Proceedings of Fourth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2002), in conjunction with the Eleventh International Conference on Information and Knowledge Management (CIKM2002)*, pages 92–95, LcLean, Virginia, USA, November 8 2002.

[15] Giansalvatore Mecca and Paolo Atzeni. Cut and Paste. *Journal of Computer and System Sciences*, 58(3):453–482, 1999.