Bare Demo of IEEEtran.cls for IEEE Computer Society Journals

Michael Shell, *Member, IEEE,* John Doe, *Fellow, OSA,* and Jane Doe, *Life Fellow, IEEE*

Abstract—A sitemap represents an explicit specification of the design concept and knowledge organization of a website and is therefore considered as the website's basic ontology. They do not only present the main usage flows for users, but also hierarchically organize concepts of the website. Typically, sitemaps are defined by webmasters at the very initial stages of the website's design. However, during their life, websites significantly change in their structure, in their content and in their possible navigation paths. Even if this is not the case, webmasters can fail in either defining sitemaps that reflect the actual website content or, vice versa, in defining the actual organization of pages and links which do not reflect the intended organization of the content coded in the sitemaps. In this paper we propose a novel approach which automatically generates sitemaps. On the contrary of other approaches proposed in the literature, which mainly generate sitemaps from the textual content of the pages, in this work sitemaps are generated by analyzing the web graph of a website. This allows us to: *i)* automatically generate a sitemap on the basis of possible navigation paths, *ii)* compare the generated sitemaps with either the sitemap provided by web designer or with the intended sitemap of the website and, consequently, *iii)* plan possible website re-organization. The solution we propose is based on closed frequent sequences extraction and is able to only concentrate on hyperlinks organized in "web lists", which we intend as logical lists embedded in the pages (which include menus, navbars and content tables).

Index Terms—Computer Society, IEEE, IEEEtran, journal, LaTeX, paper, template.

1 Introduction

One of the recognized problems in Web Mining concerns the automatic construction of web pages hierarchies, typically called *sitemaps*. A sitemap represents an explicit specification of the design concept codified by User Experience Designers and Information Architects to define the knowledge organization of a website through grouping of related content [1]. This hierarchical organization of the content is coherent with the approach typically followed in the website navigation, where users start with the homepage or a web page found through a search engine or linked from another website, and then use the *navigation systems*[1] provided by the website to find the desired information [3].

Sitemaps help *users* by: *i)* increasing the user experience of a website; *ii)* providing a complementary tool to the *keyword-based search* for the information retrieval process. In the first case, sitemaps organizing the website content in a top-down fashion, from a more general page to a detailed page, help users to understand, at glance, contextual information such as relationships among web pages (e.g. relationships of super/sub category), facilitates the discovery and the access of information and creates a better sense of orientation. In the second case, sitemaps help users when they do not know what they are looking for until the available options are presented or their information needs cannot be formulated in keywords [4], [5]. In fact, in the keyword-based paradigm the search engine, given one or several key terms, returns an ordered list of web pages in descending order of relevance and accuracy. However, users have to express their information needs in form of keyword sequences which should be as much discriminative as possible. For example, a sitemap can be useful in an university website to look for all professors or research areas. This kind of query is hard to answer by a search engine, but very simple to extract from a well designed sitemap.

Moreover, sitemaps help *search engine bots* to extract the information asset of an organization, as it is available on its website, without access to the underlying organization's databases. For example, given an organization website, it is possible to discover all its affiliate companies, products, employees, etc. Finally, a sitemap can be used, in addition to the motivations reported before, also to improve existing applications like search engines integrating taxonomies in the presentation of search results [6], or to cluster web pages having same semantic type (e.g. web pages related to professors, courses, books, lists of publications of a single researchers) [7].

The sitemap construction is not a simple process, especially for websites with a great amount of content and with wide and deep logical hierarchies. Before the introduction of the Google Sitemap Protocol (2005), sitemaps were mostly manually generated. However, as websites got bigger, it became difficult for web-masters to keep the sitemap page updated (e.g. by inserting, removing pages or adding new sections in the website), as well as list all the pages and all the contents in community-building tools like forums, blogs and message boards. This means that manually generated sitemaps could do not describe the correct and current structure of the website, becoming soon helpless and confusing for users. As regards search engines, they cannot keep track of all this material, skipping information as

- *M. Shell was with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: see http://www.michaelshell.org/contact.html*
- *J. Doe and J. Doe are with Anonymous University.*

Manuscript received April 19, 2005; revised August 26, 2015.

1. A navigation system is a common set of hyperlinks implemented using similar layout which assist users during website's navigation (e.g. navbars, menus, product lists, etc.) [2].

they crawl through changing websites. To solve this issue several automatic tools were proposed on the Web [2]. These services generate an XML sitemap, used by search bots, which enumerate a *flat* list of urls and do not output the hierarchical structure of websites.

Automatic generation of *(hierarchical)* sitemaps solves this problem, helping both the web designer to track evolutions in the website's hierarchy and users to have always updated views of the content of the website. Moreover, analyzing the web log files and comparing them with the real sitemap of a website, it is possible to understand if users browse the website in ways that are different from the designer's expectations and view the website link structure differently from the designer [8].

Several works face the problem of the automatic extraction or generation of sitemaps (also called hierarchies or taxonomies). They are usually based on analysis of text, hyperlinks, urls structure, heuristics or a combination of these features [9], [10], [11], [12]. The most prominent works, however, are mainly based on the textual content of the web pages. This approach, although proved to be effective in the generation of reasonable sitemaps, turn to be ineffective in at least two cases: *i)* when there is not enough information in the text of a page; *ii)* when web pages have different content, but actually refer to the same semantic class. The former case refers to web pages of poor of textual information, such as pages rich of structural data (e.g. pages from Deep Web Databases) or multimedia data, or when web pages have several script terms, which can be easily found also in other pages (e.g. pages from a CMS website). The latter case refers to web pages having the same semantic type but characterized by different distribution of terms. In this case, it is hard to organize web pages of the same type in the same branch of the sitemap. This aspect can be explained with an example: consider a Computer Science department website. Sitemaps generated on the basis of the textual content can be led to organize the web page of a *professor* as a child of its *research area* web page rather than as a child of the *professors* web page. In this way, web pages clustered together as siblings in the hierarchy by web masters are split in different parts of the extracted hierarchy (i.e. different research areas).

This paper is intended to be a contribution in the direction of extracting sitemaps automatically by combining information on web page structure and the hyperlink structure of websites. This goal is achieved by analyzing web pages' HTML formatting (i.e. HTML tags and visual information rendered by a web browser) to extract from each page collections of links, called *web lists*, a compact and noise-free representation of the website's graph. Then, the extracted hyperlink structure is used to study the reachability properties in the website's graph.

In order to consider reachability, the solution we propose exploits the random walk theory and the concept of frequent sequences of web pages in random walks. The basic idea is that of Distributional hypothesis, initially defined for words in natural language processing (i.e. *"You shall know a word by the company it keeps"*) [**?**] and recently extended to generic objects [**?**]. In our context we translate this idea in *"You shall know a web page by the paths it keeps"*. According to this hypothesis, two web pages are sibling in the hierarchy if they share the same most frequent path from the homepage in the website's graph.

The algorithm we propose, named *SMAP* alternates a four-steps strategy that: 1. generates the web graph using web-lists, 2. extracts sequences which represent navigation paths by exploiting random walk theory, 3. mines frequent closed sequential patterns through the application of an algorithm for sequential pattern mining, and 4. prunes discovered patterns to extract the sitemap in the form of an hierarchy. As anticipated before, on the contrary of other approaches for sitemap generation that use the hyperlink structure, SMAP uses also the web page structure (i.e. web lists), which allow us to concentrate on a subset of links which describe website's navigational systems. Moreover, SMAP uses the concept of frequent paths to provide statistical support to the structure of the generated sitemap.

This paper is organized as follows:

2 Related Work

The motivation for this work comes from research reported in the literature on sitemap extraction, application of sequential pattern mining algorithms in the context of web mining and automatic extraction of web lists. In the following, we discuss related work in these three research fiedls.

2.1 Sitemap Extraction

The problem of generating website hierarchies is part of a more the general research topic of Web Mining. Although extracting websites sitemaps is an important task for many web applications, few studies specifically focus on this problem. In the following we revise some related works done in web mining research area, even if not directly related to the specific task of sitemaps generation. We classify them on the basis of the input data the proposed methods use.

The *first category* includes algorithms of Web Content Mining which take as input a collection of textual documents. Hierarchies obtained by these methods, which typically simply perform hierarchical clustering, represent the topical organization of web pages (see [13] for a survey). However, most of the existing techniques assume that web pages share consistent writing styles, provide enough contextual information, are plain and completely unstructured, and are independent and identically distributed. The problem with hierarchy induction only based on text is that words often have multiple meanings and for heterogeneous websites it is difficult disambiguate words and construct the proper taxonomy. This is especially true for web pages poor of textual information or web pages written in collaborative manner and then having different distribution of terms. Differently from traditional textual documents, web pages are characterized by structural features, such as the hyperlinks or the HTML structure which provide different and complementary information.

The *second category* includes algorithms of Web Structure Mining which take as input a graph where nodes represent web pages and edges represent hyperlinks. Here, researchers and practitioners have used the hyperlink structure to organize web pages for many years. The basic idea

2. http://slickplan.com/, http://www.screamingfrog.co.uk, https://www.xml-sitemaps.com/

of web structure mining algorithms is that if there is a hyperlink between two pages, then some semantic relation may exist between them [**?**]. A web structure mining naïve solution for sitemap generation is the application of the simple breadth search algorithm. In the breadth search, starting from the homepage, links can be traversed level-wise and each webpage is put onto a conceptual level of the first time it is encountered. In this way, the hierarchy represents the shortest path from the homepage to each page. The problem with this method is that the shortest path from the homepage to a page does not necessarily match super/sub category relationships among pages in the path. This is due by the presence of short-cut links which connect web pages belonging to deeper levels of the hierarchy from shallow levels.

A more sophisticated approach is presented in [11], where the authors present a system based on the HITS algorithm for the automatic generation of hierarchical sitemaps from websites. The idea is to split web pages into blocks and identify those having high frequency and hub value which describe the sitemap. The accuracy of the method strongly depends on the task of blocks extraction. In fact, the algorithm requires that blocks have the same structure on the web pages. This assumption holds for blocks which are part of the navigation system at the web pages which are firstly accessed by the user and typically represent the main content of the web site, but it is not true for more specific and pages at deeper levels of the website. In fact, many websites change the the layout of secondary menus in deeper web pages to provide users a sense of progression through the website. For this type of websites the proposed algorithm may fail in properly identifying nodes of the sitemaps which correspond to pages at deeper levels of the web site. Another drawback is due by the presence of false positive blocks that are included because recurrent, albeit not belonging to the navigation system (e.g., advertisements).

In [14] the authors focus on the problem of extraction structured data such as menus to identify the main hierarchical structure and website boundaries analyzing cliques among in the web graph. As in [11], the algorithm segments web pages of a given website in blocks and identifies blocks that compose the maximal cliques as main menus of the website. Although the method is unsupervised and then suitable for the analysis of heterogeneous websites, it may suffer from the same limitations of [11] when processing deeper levels of the website.

Other state-of-art algorithms combine the hyperlink structure with content information of web pages to extract the hierarchical organization of a website. In [10] a classifier is learned to extract topic-hierarchies using several features such as URL structure, content and navigation features, information about anchors, text and heuristics. Although the method is simple and results seem to be enough accurate, labeling examples requires a lot of effort. Therefore, the solution is not directly applicable in huge and heterogeneous websites. In [12] a method that extracts document-topic hierarchies from websites is proposed. This method combines information from text and hyperlinks by alternating two steps: 1) Random walking with restart from homepage and assigning an initial multinomial distribution to each node (i.e., document); 2) updating the multinomial distribution when the walker reaches a node. These steps are realized using Gibbs sampling algorithm. Several thousand Gibbs-iterations are usually required before a hierarchy emerges. The resulting hierarchy is obtained selecting, for each node, the most probable parent, in the way that higher nodes in the hierarchy contain more general terms, and lower nodes in the hierarchy contain more specific terms.

2.2 Sequential pattern mining for web mining

This work has also connections with works in the Web Usage Mining field. Goal of these approaches is to apply sequential pattern mining algorithms for identifying patterns in web log files which describe how users navigate the website [15], [16]. In general, web usage mining algorithms which extract hierarchies based on user behaviors analyze two different types of patterns: *sequential patterns* and *contiguous sequential patterns* [16]. Sequential patterns are sequences of items that frequently occur in a sufficiently large proportion of transactions, while contiguous sequential patterns are a special form of sequential patterns in which the items appearing in the sequence must be adjacent with respect to the underlying ordering followed by the users during navigation.

In [17] the authors compare models based on sequential and contiguous sequential patterns for predicting the next page that the user will navigate. They claim that models based on contiguous sequences better perform on websites with deeper structure and longer paths, while prediction models based on sequential patterns are better suited for personalization in websites with a higher degree of connectivity and shorter navigation depth (i.e., shallow hierarchies). However, at the best of our knowledge, there is no study that analyzes these models in the context of sitemap generation, via hyperlink analysis. In any case, algorithms for the generation of web page hierarchies based on users' behavior cannot be directly applied to sitemap generation because they typically create multiple hierarchies (i.e., one for each user profile) and only consider web pages having a user-specified number of visitors.

2.3 Automatic extraction of web lists

As claimed in [2], not all the links are equally important to describe the website structure. Several works in the field of Web Mining exploit web pages taking the advantage of the structural and visual information embedded in HTML tags. In [2], [7], [18] collections of hyperlinks having similar visual and/or structural properties can be used to filter noisy links and collect web pages belonging to same semantic type.

In this research line, [2], [7] identify web lists and exploit them for the task of web page clustering. Specifically, in [2] the authors rely on the layout properties of link collections available in the pages (they use the set of paths between the root of the page and the HTML tags $< a >$ to characterize the structure) to find structurally-similar web pages. In [7] the authors propose a similarity measure obtained combining textual similarity, co-citation and bibliography-coupling similarity and in-page link-structure similarity. In this way, two web pages have a similar in-page link-structure if they frequently appear together in link collections. Differently, in [18] the authors define the concept of logical web lists (i.e.

web lists that collect structured data spanned in multiple pages of the same website) for information extraction purposes.

We follow this basic idea, we use visual and/or structural properties for the identification of the most promising nodes for sitemap extraction. The aim is to codify, in this way, the navigation systems of a website and, accordingly, reduce the search space during sitemap extraction.

3 Extraction of Sitemaps: Preliminary Definitions

Before describing the proposed methodology for automatic sitemap extraction, we provide some formal definitions. Such definitions characterize the web page structure and the hyperlink structure of web pages.

A web page is characterized by multiple representations, such as a textual representation (composed by web page terms), a visual representation (composed by information about rendered web page) and a structural representation (composed by HTML tags). Our method takes into account both the visual and the structural representations. For this reason, we provide in the following formal definitions which identify the sources of information we exploit.

Definition 1. *A web page is characterized by a **Structural Representation** composed by web elements inscribed in HTML tags and organized in a tree-based structure. HTML tags can be applied to pieces of text, hyperlinks and multimedia data to give them different meaning and rendering in the web page.*

Definition 2. ***Web Page Visual representation.*** *When a web page is rendered in a web browser, the CSS2 visual formatting model [19] represents the web page's elements by rectangular boxes that are laid out one after the other or nested inside each other by forming a tree, called **Rendered Box Tree**. By associating the web page with a coordinate system whose origin is at the top-left corner, the spatial position of each web page's element is fully determined by the tuple (x, y, h, w), where (x, y) are the coordinates of the top-left corner of its corresponding box (i.e. the position of the box in the rendered page), and (h, w) are the box's height and width respectively (i.e. the size of the box in the rendered page). Therefore, the **Visual Representation** of a web page is given by its Rendered Box Tree.*

The rendered box tree can be generated by any web browser which follows W3C specifications for rendering [19]. Moreover, the rendered box tree could have a completely different structure from that of the corresponding HTML tag tree. This because *i)* a web page can be enriched of invisible elements (like the <head> tag or elements that have *display:none;* set), and *ii*) the generation of the rendered box tree requires the execution of javascript and css code.

Figure 1 shows an example of the structural and visual representations for the homepage of a Computer Science Department website. Leafs of the structural tree and the Rendered Block Tree represent the minimum semantic units that cannot be segmented further (e.g., images, plain texts, links). Actually, the Rendered Block Tree is more complicated than what Figure 1(d) shows (there are often hundreds or even thousands of blocks in a Rendered Block Tree).

Both Definition 1 and Definition 2, which are used to exploit the web page stucture, are used in the definition of web lists, which is crucial for our approach:

Definition 3. *A **Web List** is a collection of two or more web elements (called data records) codified as rendered boxes having a similar HTML structure, and visually adjacent and aligned. This alignment can occur via the x-axis (i.e. a vertical list), the y-axis (i.e. horizontal list), or in a tiled manner (i.e. aligned vertically and horizontally) [18].*

This definition requires an algorithm which is charge of checking whether two rendered boxes "have a similar HTML structure". Moreover, it also requires a formal definition of alignment and adjacency. These aspects are discussed in Section 4.

In Figure 1(b) we report the example of a Computer Science Department website. From the reported page, one horizontal web list (box A) an three vertical web lists (boxes B, C, D) can be extracted.

Another important source of information our approach uses is the hyperlink structure of the website, which is formally introduced by the following definition:

Definition 4. *A **Website** is a directed graph $G = (V, E)$, where V is the set of web pages and E is the set of hyperlinks. In most cases, the homepage h of a website represents the website's entry page and, thus, allows the website to be viewed as a rooted directed graph.*

As anticipated, our algorithm for the automatic identification of sitemaps exploits a sequential pattern mining step. The main rationale behind this choice is to see a navigation path in a website as a sequence of urls and use sequences of urls to identify common (and frequent) navigation paths. Formally, a sequence is defined as follows:

Definition 5. ***Sequence**: Let $G = (V, E)$ be a website, a sequence S is defined as $S = \langle t_1, t_2, \ldots, t_m \rangle$, where each item $t_j \in V$ denotes the web page at the j-th step in the navigation path S.*

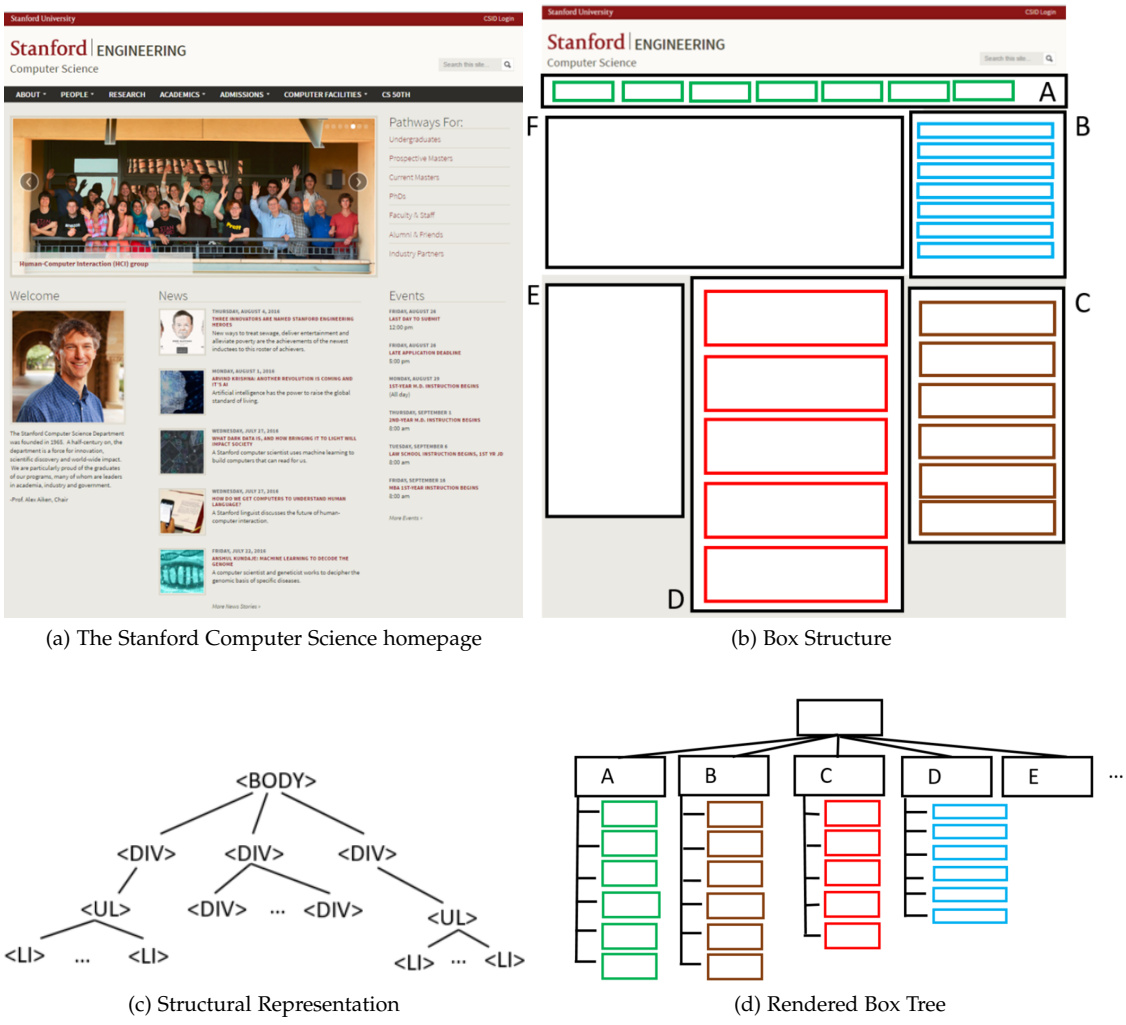
A sequence $S = \langle t_1, t_2, \ldots, t_m \rangle$ is a **sub-sequence** of a sequence $S' = \langle a_1, a_2, \ldots, a_n \rangle$, if and only if integers $i_1, i_2, \ldots, i_m$ exist, such that $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ and $t_1 = a_{i1}, t_2 = a_{i2}, \ldots t_m = a_{im}$. We say that S' is a **super-sequence** of S or that S' contains S.

Example 1. *The sequence $S' = \langle a, b, d \rangle$ is a super-sequence of $S = \langle a, d \rangle$. On the contrary, S' is not a super-sequence of $S'' = \langle e \rangle$, since the item e is not equal to any item of S'.*

Given a database of sequences SDB and a single sequence S the *absolute support* of S in SDB is the number of sequences in SDB which contain S, while its *relative support* is the absolute support divided by the size of database (i.e., $|SDB|$). With the term support will refer to the relative support, unless otherwise specified:

$$\sigma(S) = \frac{|\{t \in SDB : S \text{ is a sub-sequence of } t\}|}{|SDB|} \quad (1)$$

A sequence is frequent if its support is greater than a user defined threshold. In our case the support defines the relationship strength among web pages.



(a) The Stanford Computer Science homepage

(b) Box Structure

(c) Structural Representation

(d) Rendered Box Tree

Fig. 1

Following the suggestion provided in [16], we also exploit the definition of contiguous sequence which is formally defined as follows:

Definition 6. ***Contiguous Sequence****: Given a sequence $S = \langle t_1, t_2, \ldots, t_m \rangle$ created from $G = (V, E)$, S is contiguous if each item t_i appears in G contiguously to item t_{i-1} (i.e. there is an edge between t_{i-1} and t_i).*

The task of sequential pattern mining indicates the task of discovering frequent (sub-)sequences (contiguous sequences) from SDB. This is considered computationally challenging because algorithms have to generate and/or test a combinatorially explosive number of intermediate subsequences. For this reason, we consider the simpler problem (in terms of time complexity) of identifying *closed* sequential patterns which are compact and provide a lossless representation of all sequential patterns. A closed sequential pattern (or, simply, a closed sequence) is defined as follows:

Definition 7. ***Closed Sequence****: A sequential pattern (contiguous sequential pattern) S_j is* ***closed*** *if and only if its support is different from all of its frequent super-sequences.*

Example 2. *Figure 3(b) shows an example of a database of sequences. Sequence $\langle h \rangle$ is closed because no super-sequences of $\langle h \rangle$ with the same support exist. On the contrary, the sequence $\langle h, d \rangle$ is not closed because the super-sequence $\langle h, b, d \rangle$ has the same support of $\langle h, d \rangle$.*

The last definition we have to provide before discussing technical details of the method is that of sitemap. This definition is particularly important since it defines the goal of the proposed method.

Definition 8. ***Sitemap****: Given a web graph $G = (V, E)$ where $h \in V$ is the homepage, a user-defined threshold t and a weight function $w : E \rightarrow \mathbb{R}$, then $T = \arg\max_{T_i} \Phi(T_i)$ is a sitemap if:*

1) *$T_i = (V_i', E_i')$ is a tree rooted in h, where $V_i' \subseteq V$ and $E_i' \subseteq E$;*
2) *$\Phi(T_i) = \sum_{e \in E_i'} w(e)$;*
3) *$\forall\, e = (j_1, j_2) \in E_i', j_2 \in webList(j_1)$, that is, the url of the web page j_2 is contained in some web list of the web page j_1 (See Def. 3);*
4) *$\forall e \in E_i',\ w(e) \geq t$.*

What remains unspecified in this (general) definition is the meaning of $w(\cdot)$ and the way the tree T is generated.

These aspects are discussed in the following section where we describe the method and instantiate aspects intentionally kept as parameters in Definition 8. We only anticipate that our solution does not need to enumerate all possible trees $T_i \subseteq G$ to optimize $\Phi(\cdot)$, yet it is able to extract directly T analyzing a sub-graph of the website.

4 METHODOLOGY

The problem we consider is that extracting the website's sitemap, defined according to Definition 8. To achieve this goal, we combine different techniques which come from web and data mining background: *i)* Information extraction for identifying potential navigation systems in web pages in form of web-lists; *ii)* Random Walk theory to extract navigation paths; *iii)* Sequential pattern mining for finding the most frequent paths which describe the hierarchical structure of the website.

In the first step of the proposed methodology we perform website crawling. Crawling uses web pages' structural and visual information to extract web lists in order to mitigate problems coming from noisy links. The output of this phase is the website graph G, where each node represents a single page and edges represent hyperlinks. In the second phase we generate sequences of urls, that is, navigation paths. This is done by exploiting random walks extracted from the crawled website's graph. In the third phase closed sequential patterns are mined. The output of this phase is a tree of frequent sequences. The last step is in charge of generating a sitemap by means of a pruning strategy. This pruning strategy takes as input the tree of frequent sequences and extracts a tree T (sitemap) where the homepage is the root and a web page appears only once, that is, it is not possible to have multiple paths that connect the homepage with a web page.

4.1 Website crawling

As claimed in [2] not all links are equally important to describe the website structure. In fact, a website is rich of noisy links, which may not be relevant to sitemap extraction process, such as hyperlinks used to enforce the web page authority in a link-based ranking scenario, short-cut hyperlinks, etc. Moreover, as clarified in Section 1 the website structure is codified in navigational systems which provide a local view of the website organization. In this respect, the solution we propose, based on the usage of web lists, has a twofold effect: from one side it guarantees that only hyperlinks which may belong to potential navigation systems are considered; on the other side, it allows the method to implicitly take into account the web page structure which is implicitly codified in the web lists available in web pages [?], [?], [2], [18].

The crawling algorithm is described in Algorithm 1. In particular, starting from the homepage h, the method *extractWebLists()* is iteratively applied to extract url collections having the same domain of h and organized in *web lists*. Only urls (i.e. web pages) included in web lists are further explored (line 6). The output of website crawling step is the graph $G = (V, E)$ which will be used in the next step.

Concerning Definition 3, to check whether two web elements e_i, e_j have similar HTML structure, we first codify the HTML tree having as root the web element e_i (e_j) in a string composed by HTML tags ordered applying a breadth search on the rooted tree; then, we apply to the generated strings the *Normalized Edit Distance*. Normalized edit distance is suitable to extract web lists whose data records have a simple and a very similar HTML structure (such as most web lists describing websites' navigation systems). To discover more complex web lists, in terms of data records' HTML structure, the normalized edit distance can be replaced by the normalized *tree edit distance* [20]. Similar to edit distance between strings, the tree edit distance between two trees A and B is the cost associated with the minimum set of operations needed to transform A into B. Since for the analyzed websites the Normalized Edit Distance already works very well, we just used the normalized edit distance between strings for the task of web list extraction.

Two web elements e_i, e_j are adjacent if they are sibling in the Rendered Box Tree and there is no a web element e_w, with $e_w \neq e_i \neq e_j$, which is rendered between them. Finally, two web elements e_i, e_j are aligned on: *i)* the x-axis if the have same x coordinate, that is $e_i.x = e_j.x$; *ii)* the y-axis if they share the same y value, that is $e_i.y = e_j.y$; *iii)* both axes, that is $e_i.x = e_j.x$ and $e_i.y = e_j.y$.

Algorithm 2 describes in details the web list extraction process. In particular the algorithm analyzes all rendered web element which are descendant of *body tag* element. For each *element* to analyze (line 4) if the dimension of the tree, rooted at the *element*, is lower than a threshold *maxSize* (line 6) the algorithm tries to align its children (lines 9-10). In our experiments, we set *maxSize* to 30 which is a sufficient threshold to find navigation systems. Only the web elements which are included neither in *verticalLists* nor in *horizontalLists* are further explored by the algorithm (lines 17-18).

Algorithm 1 crawlingWebsite(homepage)

Input: URL homepage;
Output: Set<(URL, URL)> E; Set<URL> V;
1: frontier = Set()
2: Q = Queue(homepage)
3: **repeat**
4: currentPage = Q. *dequeue*();
5: V. *add*(currentPage);
6: webLists = *extractWebLists*(currentPage) . *filterSameDomain*(homepage);
7: **for each** list ∈ webLists **do**
8: pagesToAnalyze = list. *toSet*(-); frontier;
9: Q. *enqueue*(pagesToAnalyze);
10: frontier. *add*(pagesToAnalyze);
11: **for each** u ∈ pagesToAnalyze **do**
12: E. *add*((currentPage, u));
13: **end for**
14: **end for**
15: **until** !queue.empty()
16: **return** (V, E)

4.2 Sequence Database Generation

or, equivalently, a navigation path

Algorithm 2 extractWebLists(webpage)

Input: URL webpage;
Output: List<List<URL>> W;
1: body = *findElementByTagName*("body", webpage);
2: notAligned = body. *getChildren*();
3: **repeat**
4: element = notAligned. *dequeue*();
5: children = element. *getChildren*();
6: **if** element.getSize() >= maxSize **then**
7: notAligned. *enqueue*(children);
8: **else**
9: verticalLists = children.groupBy(c → c.x) .map(list → *getStructurallySimilar*(list)) .filter(list → list.size ≥ 2)
10: horizontalLists = children.groupBy(c → c.y) .map(list → *getStructurallySimilar*(list)) .filter(list → list.size ≥ 2)
11: **for each** list ∈ verticalLists **do**
12: W.add(new List(list. *getUrls*()))
13: **end for**
14: **for each** list ∈ horizontalLists **do**
15: W.add(new List(list. *getUrls*()))
16: **end for**
17: notIncluded = *getNotAligned*(children, verticalLists, horizontalLists);
18: notAligned. *enqueue*(notIncluded);
19: **end if**
20: **until** !notAligned.empty()
21: **return** W

To capture and codify correlations among graph's nodes (i.e. web pages) we use the Random Walk with Restart from Homepage (RWRH) approach. It can be considered a particular case of Random Walk with Restart, obtained setting a single node (i.e. the homepage) as starting vertex. Random walk with restart is widely adopted in several works to infer structural properties of nodes in a graph through the analysis of the global structure of the whole network.

Using this approach, web pages closer to the homepage have higher probability to be reached than those at deeper levels. This is coherent with the organization typically followed in the website navigation where navigation systems closer to the homepage belong to shallower levels of the website hierarchy.

Therefore, given the web graph G extracted in the previous step, and two number $rwrLength, dbLength \in \mathbb{N}$, the random walk theory is used to navigate the web graph G from the homepage h. The output of this phase is a sequence database SDB composed by $dbLength$ random walks having length $rwrLength$ and starting from h. As defined in [21], increasing the length $rwrLength$ of a random walk starting at node i, the probability to reach a node j tends to depend on the degree of j. As suggested in [21] we generate short random walks in the way that the random walker tends to get trapped into densely connected parts of the graph corresponding to communities (i.e. sibling pages in the sitemap) and for avoiding to infer false correlations among web pages due the nodes' degree. Algorithm 3

Algorithm 3 rwrGeneration(rwrLength, dbLength, G, α)

Input: int rwrLength, int dbLength, Graph G, float α;
Output: List<List<URL>> randomWalks;
1: **for each** i ∈ Range(0, dbLength) **do**
2: w = List()
3: w[0] = G.homepage
4: **for each** j ∈ Range(1, rwrLength) **do**
5: λ = Math.random()
6: **if** $\lambda > \alpha$ **then**
7: w[j] = G. *getRandomOutlink*(w[j-1]);
8: **else**
9: w[j] = w[0]
10: **end if**
11: **end for**
12: randomWalks. *add*(w);
13: **end for**
14: **return** randomWalks

describes the generation process of the sequence dataset. Figure 3b shows a sequence database obtained from the web graph in Figure 3a

4.3 Closed sequential patter mining

Given the sequence database (SDB), extracted in the Section 4.2, and a user defined threshold t (i.e. minimum support), we apply a closed sequential pattern mining algorithm to extract all the frequent sequences. In this phase we used the algorithm CloFAST [22] as closed sequential pattern mining algorithm. It showed to provide compact results, saving computational time and space if compared with other closed sequential pattern mining algorithms. CloFast returns a tree $T_{CloFAST} = (V', E')$ and a weight function $\sigma : E' \rightarrow \mathbb{R}$ which associates each edge $e = (j_1, j_2) \in E'$ with the *relative* support of the sequence $\langle h, \ldots, j_1, j_2 \rangle$ in $T_{CloFAST}$.

Figure **??** shows an example of tree extracted by CloFAST for the input sequence dataset in figure **??**.

Since a contribution of this paper is compare the sitemaps extracted considering two different types of sequences (i.e. sequential patterns and contiguous sequential pattern), we extend CloFAST by allowing it to also extract contiuous sequential patterns.

To extract the weighted tree of contiguous sequences from $T_{CloFAST}$ we benefit of the data structure, called *Vertical id-List* (VIL) provided by CloFAST for support counting purpose and for extracting frequent sequences. In the following we give a brief definition of a VIL:

Definition 9 (Vertical Id-list)**.** *Let SDB be a sequence database of size* n *(i.e. $|SDB| = n$), $S_j \in SDB$ its j-th sequence ($j \in \{1, 2, \ldots, n\}$), and α a sequence associated to a node of the tree, its* vertical id-list*, denoted as VIL_α, is a vector of size n, such that for each $j = 1, \ldots, n$*

$$VIL_\alpha[j] = \begin{cases} [pos_{\alpha,1}, pos_{\alpha,2}, \ldots, pos_{\alpha,m}] & \text{if } S_j \text{ contains } \alpha \\ null & \text{otherwise} \end{cases}$$

where $pos_{\alpha,i}$ is the end position of the i-th occurrence ($i \leq m$) of α in S_j.

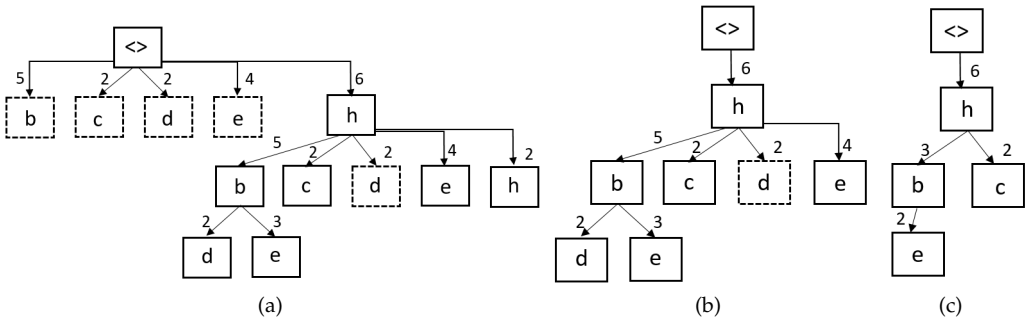
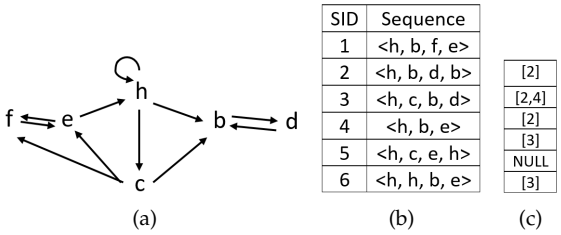


Fig. 2: (a) Frequent sequence tree extracted by CloFAST with absolute minsup = 3 and SDB in fig. 3(b). Nodes with dashed borders represent non-closed nodes; (b) Frequent sequences tree extracted by extended version of CloFAST and having the support as weight function; (c) Contiguous sequences tree extracted by extended version of CloFAST and having the contiguous support as weight function.



SID	Sequence
1	<h, b, f, e>
2	<h, b, d, b>
3	<h, c, b, d>
4	<h, b, e>
5	<h, c, e, h>
6	<h, h, b, e>

[2]
[2,4]
[2]
[3]
NULL
[3]

Fig. 3: (a) Web Graph rooted at h; (b) Sequence Database (SDB), that is, a set of tuples (SID, Sequence), where SID is a sequence-id and Sequence is a random walks with restart from the homepage; (c) VIL for the sequence $\alpha = \langle h, b \rangle$.

Example 3. *Figure* **??** *shows the VIL of the sequence $\alpha = \langle A, B \rangle$. Values in VIL_α represent the end position of the occurrences of the sequence α in the sequences of Figure 3b. In particular, the first element (list with only value 2) represents the position of the first occurrence of web page B, after the web page A (i.e. B is the last item in α), in the first sequence S_1. The second element is (list with only value 3) the position of the first item B (after A) in the sequence S_2. The third element is null since α is not present in S_3. The other values are respectively list with only value 2 (for sequence S_4), list with only value 3 (for S_5),* null *(for S_6) and list with only value 3 (for S_7). Moreover, the support of α is the number of not null elements in VIL_α, that is 0.71.*

We modify the CloFAST implementation creating two pruning method that: *i)* stop the generation of frequent sequences which start from a node different from the homepage; *ii)* extract contiguous sequences. In particular, starting from the root of $T_{CloFAST}$, the function *contiguous(·)* (see Algorithm 4) is iteratively applied to update the node's VILs. It looks for possible holes in the sequences by botton-up clibing the sequence tree $T_{CloFAST}$. An hole is found when the condition at line 10 is not satisfied. Returned VIL of a node n is used to calculate its contiguous support without scanning the sequence database SDB: $\sigma_c(n) = \{j | vil = contiguous(n, T_{CloFAST}) \wedge vil[j][1] = h\}$. Therefore, if the contiguous support of n, $\sigma_c(n)$, is greater of the threshold t, the function is applied to its children, otherwise the node is pruned. We call the returned tree $T'_{CloFAST}$

4.4 Sequence Pruning

The trees $T_{CloFAST}$ (for sequential patterns) and $T'_{CloFAST}$ (for contiguous sequential patterns), extracted in the previous step, may contain multiple frequent paths to reach, from the homepage, a given node (web page). For example, in Figure **??** the web page c can be reached using the frequent patterns $h \to a \to c$ and $h \to b \to c$. Dire che quindi l'albero di CloFAST corrisponde ad un grafo. Since in the sitemap each web page can be reached by only one path starting from the homepage, the last task of the sitemap generation is the pruning process. Therefore, the goal of this step is to select for each web page included in the frequent sequence (contiguous sequence) tree, the best path to reach it.

A sequence $\alpha = \langle u_1, \ldots, u_i, u_{i+1}, \ldots, u_n \rangle \in T_{CloFAST}$ is pruned if:

1) $\nexists\, (u_i, u_{i+1}) \in E$, with $1 \leq i < n$ and $u_1 = h$;
2) $\exists \beta = \langle h, \ldots, u_n \rangle \in T_{CloFAST}$ ($T'_{CloFAST}$) and $\beta \neq \alpha$ such that

$$\sum_{e' \in T_\beta} w(e') + w(\beta) > \sum_{e \in T_\alpha} w(e) + w(\alpha) \qquad (2)$$

The first constraint ensures that frequent paths that do not exist in the crawled graph G are removed (see Def. 8.3). The second constraint allows us to select for each node in V' which can be reached by multiple frequent sequences, starting from the homepage, the best path. In particular, for each node u_n, ending node of multiple frequent sequences (e.g. α and β), we compare all the sub-trees of $T_{CloFAST}$ ($T'_{CloFAST}$) having u_n as root (e.g. T_α and T_β). From them, we only keep that one which maximize the sum of its edges plus the weight of the path starting from h and reaching u_n (e.g. $w(\alpha)$ and $w(\beta)$). For the case of sitemap extraction using sequential patterns we associate to the weight function $w(\cdot)$, defined in Def. 8, the support function $\sigma(\cdot)$ returned by CloFAST. For the case of sitemap extraction using contiguous sequential patterns we associate to $w(\cdot)$ the contiguous support function $\sigma_c(\cdot)$ calculated by Algorithm 4.

5 Experiments and Discussion

For HDTM we set $\gamma = 0.25$. This value was chosen via experimentation to determine the best possible hierarchy in

Algorithm 4 contiguous(T,n)

Input: T: a sequence tree extracted by CloFAST; n: node of T

Output: vil: the VIL of the sequence at node n such that the contiguous condition is satisfied.

```
1: vil = getVil(n);
2: parent = getParent(n);
3: repeat
4:    parentVil = getVil(parent);
5:    for all j = 1 ... length(vil); do
6:       i=0; contiguous = FALSE;
7:       repeat
8:          z=0;
9:          repeat
10:            if vil[j][i] = parentVil[j][z]+1 then
11:               vil[j] = parentVil[j][z..(len(parentVil[j])-1)];
12:               contiguous = TRUE ;
13:            end if
14:         until ++z ≤ i AND !contiguous
15:         if !contiguous then
16:            vil[j] = NULL
17:         end if
18:      until ++i < len(vil[j]) AND !contiguous
19:   end for
20:   n = parent
21:   parent = getParent(n);
22: until parent != root(T)
23: return vil;
```

terms of precision and recall. We analyzed the effectiveness of proposed approach (SMAP) with four real websites: *cs.illinois.edu*, *www.cs.ox.ac.uk*, *www.cs.princeton.edu*, and *www.enel.it*. Each analyzed website contains a sitemap provided by the website's organization which allows us to have a ground truth and evaluate the extracted sitemaps in terms of precision, recall and F-measure of edges. The results are compared with the hierarchies of web pages extracted by HDTM [12]. Table 1 shows the effectiveness of SMAP varying the minimum support: when the minimum support threshold decreases, we are able to obtain deeper and wider sitemaps. Moreover, as expected, by reducing the minimum support threshold, precision increases and recall decreases. What is interesting is that F-measure depends on the website and, more precisely, on the level of details of the ground truth (real sitemaps). Interestingly, our algorithm (SMAP) significantly outperforms HDTM, independently of the website and of the minimum support threshold.

Website	Algorithm	min. supp.	Precision	Recall	F-Measure
www.cs.illinois.edu	SMAP	0.005	0.38	0.78	0.51
www.cs.illinois.edu	SMAP	0.001	0.66	0.48	0.56
www.cs.illinois.edu	SMAP	0.0005	0.66	0.33	0.44
www.cs.illinois.edu	HDTM	–	0.12	0.1	0.11
www.cs.ox.ac.uk	SMAP	0.005	0.72	0.3	0.42
www.cs.ox.ac.uk	SMAP	0.001	0.72	0.21	0.33
www.cs.ox.ac.uk	SMAP	0.0005	0.72	0.21	0.33
www.cs.ox.ac.uk	HDTM	–	0.37	0.15	0.21
www.cs.princeton.edu	SMAP	0.005	0.61	0.55	0.58
www.cs.princeton.edu	SMAP	0.001	0.89	0.23	0.36
www.cs.princeton.edu	SMAP	0.0005	0.89	0.2	0.33
www.cs.princeton.edu	HDTM	–	0.36	0.08	0.13
www.enel.it	SMAP	0.005	0.31	0.74	0.43
www.enel.it	SMAP	0.001	0.36	0.72	0.48
www.enel.it	SMAP	0.0005	0.8	0.73	0.76
www.enel.it	HDTM	–	0.35	0.75	0.48

TABLE 1: Experimental results of SMAP and HDTM.

ACKNOWLEDGMENTS

bla bla bla

REFERENCES

[1] J. Nielsen and H. Loranger, *Prioritizing Web Usability*. Thousand Oaks, CA, USA: New Riders Publishing, 2006.

[2] V. Crescenzi, P. Merialdo, and P. Missier, "Clustering web pages based on their structure," *Data Knowl. Eng.*, vol. 54, no. 3, pp. 279–299, Sep. 2005. [Online]. Available: http://dx.doi.org/10.1016/j.datak.2004.11.004

[3] X. Fang and C. W. Holsapple, "An empirical study of web site navigation structures' impacts on web site usability," *Decision Support Systems*, vol. 43, no. 2, pp. 476 – 491, 2007, emerging Issues in Collaborative Commerce. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923606001850

[4] U. Lee, Z. Liu, and J. Cho, "Automatic identification of user goals in web search," in *Proceedings of the 14th International Conference on World Wide Web*, ser. WWW '05. New York, NY, USA: ACM, 2005, pp. 391–400. [Online]. Available: http://doi.acm.org/10.1145/1060745.1060804

[5] C. Olston and E. H. Chi, "Scenttrails: Integrating browsing and searching on the web," *ACM Trans. Comput.-Hum. Interact.*, vol. 10, no. 3, pp. 177–197, Sep. 2003. [Online]. Available: http://doi.acm.org/10.1145/937549.937550

[6] M. Keller, P. Mühlschlegel, and H. Hartenstein, "Search result presentation: Supporting post-search navigation by integration of taxonomy data," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 1269–1274. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487788.2488161

[7] C. X. Lin, Y. Yu, J. Han, and B. Liu, "Hierarchical web-page clustering via in-page and cross-page link structures," in *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, ser. PAKDD'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 222–229. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13672-6-22

[8] C. R. Anderson, P. Domingos, and D. S. Weld, "Adaptive web navigation for wireless devices," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 879–884. [Online]. Available: http://dl.acm.org/citation.cfm?id=1642194.1642211

[9] Z. Liu, W. K. Ng, and E.-P. Lim, "An automated algorithm for extracting website skeleton," in *Database Systems for Advanced Applications*. Springer, 2004, pp. 799–811.

[10] C. C. Yang and N. Liu, "Web site topic-hierarchy generation based on link structure," *J. Am. Soc. Inf. Sci. Technol.*, vol. 60, no. 3, pp. 495–508, Mar. 2009. [Online]. Available: http://dx.doi.org/10.1002/asi.v60:3

[11] S.-H. Lin, K.-P. Chu, and C.-M. Chiu, "Automatic sitemaps generation: Exploring website structures using block extraction and hyperlink analysis," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3944–3958, 2011.

[12] T. Weninger, Y. Bisk, and J. Han, "Document-topic hierarchies from document graphs," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ser. CIKM '12. New York, NY, USA: ACM, 2012, pp. 635–644. [Online]. Available: http://doi.acm.org/10.1145/2396761.2396843

[13] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms." in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer, 2012, pp. 77–128. [Online]. Available: http://dblp.uni-trier.de/db/books/collections/Mining2012.html#AggarwalZ12a

[14] M. Keller and M. Nussbaumer, "Menuminer: Revealing the information architecture of large web sites by analyzing maximal cliques," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12 Companion. New York,

NY, USA: ACM, 2012, pp. 1025–1034. [Online]. Available: http://doi.acm.org/10.1145/2187980.2188237

[15] M. Baumgarten, A. G. Büchner, S. S. Anand, M. D. Mulvenna, and J. G. Hughes, "User-driven navigation pattern discovery from internet data," in *Revised Papers from the International Workshop on Web Usage Analysis and User Profiling*, ser. WEBKDD '99. London, UK, UK: Springer-Verlag, 2000, pp. 74–91. [Online]. Available: http://dl.acm.org/citation.cfm?id=648036.744397

[16] B. Mobasher, "The adaptive web," P. Brusilovsky, A. Kobsa, and W. Nejdl, Eds. Berlin, Heidelberg: Springer-Verlag, 2007, ch. Data Mining for Web Personalization, pp. 90–135. [Online]. Available: http://dl.acm.org/citation.cfm?id=1768197.1768201

[17] M. Nakagawa and B. Mobasher, "A hybrid web personalization model based on site connectivity," in *Proceedings of WebKDD*, 2003, pp. 59–70.

[18] P. F. Lanotte, F. Fumarola, M. Ceci, A. Scarpino, M. Torelli, and D. Malerba, "Automatic extraction of logical web lists," in *Foundations of Intelligent Systems*, ser. Lecture Notes in Computer Science, T. Andreasen, H. Christiansen, J.-C. Cubero, and Z. Ra, Eds. Springer International Publishing, 2014, vol. 8502, pp. 365–374. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-08326-1_37

[19] H. W. Lie and B. Bos, *Cascading Style Sheets:Designing for the Web, 2nd Edition*. Addison-Wesley Professional, 1999.

[20] E. D. Demaine, S. Mozes, B. Rossman, and O. Weimann, "An optimal decomposition algorithm for tree edit distance," *ACM Trans. Algorithms*, vol. 6, no. 1, pp. 2:1–2:19, Dec. 2009. [Online]. Available: http://doi.acm.org/10.1145/1644015.1644017

[21] L. M. Pons, Pascal, "Computing communities in large networks using random walks." *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 191–218, 2006. [Online]. Available: http://eudml.org/doc/55419

[22] F. Fumarola, P. F. Lanotte, M. Ceci, and D. Malerba, "Clofast: closed sequential pattern mining using sparse and vertical id-lists," *Knowledge and Information Systems*, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10115-015-0884-x