World Wide Web
https://doi.org/10.1007/s11280-020-00839-2

Closed sequential pattern mining for sitemap generation

Michelangelo Ceci[1,2] · **Pasqua Fabiana Lanotte**[1]

Received: 24 July 2019 / Revised: 22 July 2020 / Accepted: 31 August 2020 /
Published online: 27 September 2020

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. It not only presents the main usage flows for users, but also hierarchically organizes concepts of the website. Typically, sitemaps are defined by webmasters in the very early stages of the website design. However, during their life websites significantly change their structure, their content and their possible navigation paths. Even if this is not the case, webmasters can fail to either define sitemaps that reflect the actual website content or, vice versa, to define 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 an approach which automatically generates sitemaps. Contrary to 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 the 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 sequence extraction and only concentrates on hyperlinks organized in "Web lists", which are logical lists embedded in the pages. These "Web lists" are typically used for supporting users in Web site navigation and they include menus, navbars and content tables. Experiments performed on three real datasets show that the extracted sitemaps are much more similar to those defined by website curators than those obtained by competitor algorithms.

Keywords Automatic extraction of sitemaps · Sequential pattern mining · Web page hierarchies · Closed patterns · Website structure mining · Data extraction and integration

✉ Michelangelo Ceci
michelangelo.ceci@uniba.it

[1] Department of Computer Science, University of Bari Aldo Moro, Bari, Italy

[2] Department of Knowledge Technologies, Jožef Stefan Institute, Ljubljana, Slovenia

1 Introduction

One of the main problems in Web mining concerns the automatic construction of Web page 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 [34]. 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. They then use the *navigation systems*[1] provided by the website to find the desired information [7].

Sitemaps help *users* by: *i)* increasing the user experience of a website and *ii)* providing a complementary tool for the *keyword-based search* in the information retrieval process. In the first case, sitemaps organize the website content in a top-down fashion, from a more general page to a detailed page, help users to have, at a glance, contextual information such as relationships among Web pages (e.g. relationships of super/sub category), facilitate the discovery of and the access to information and create 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 [24, 35]. 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 the form of keyword sequences, which should be as discriminative as possible. For example, a sitemap can be useful in a university website to identify all the professors or all the 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 accessing the underlying organization's databases. For example, given an organization website, it is possible to discover all its affiliate companies, products, employees, etc. A sitemap can also be used to improve existing applications like search engines by integrating taxonomies in the presentation of search results [18], or to cluster Web pages having the same semantic type (e.g. Web pages related to professors, courses, books, lists of publications of a single researcher) [26]. Additional applications include the extraction and integration of the semi-structured Web [41], the usage of the semantics inherent in sitemaps to learn ontologies [2], the detection of Web bots by exploiting sitemaps [29], the implementation of intelligent crawlers for archiving news websites [39], contribution-based Web page ranking [20] and automatic creation of digital libraries [16].

The sitemap construction is not a simple process, especially for websites with a large content and with wide, deep logical hierarchies. Before the introduction of the Google Sitemap Protocol (2005) sitemaps were typically manually generated. However, as websites become bigger, it became also difficult to manually keep the sitemap updated (e.g. by inserting and/or removing pages or adding new sections in the website), as well as to list all the pages and contents in community-building tools like forums, blogs and message boards. This means that manually generated sitemaps do not describe the correct, current structure of the website, soon becoming useless and confusing for users. Moreover, search engines

[1] A navigation system is a common set of hyperlinks implemented using a similar layout, which assist users during website navigation (e.g. navbars, menus, product lists, etc.) [5].

cannot keep track of all this material, skipping information as they crawl through changing websites. To solve this issue several automatic tools have been 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 hierarchy and users to have constantly updated views of the content of the website. Moreover, analyzing the Web log files and comparing them with the real sitemap of a website, makes it 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 [3].

The existing works that face the problem of the automatic extraction or generation of sitemaps (also called hierarchies or taxonomies) are usually based on the analysis of text, hyperlinks, urls structure, heuristics or a combination thereof [27, 28, 40, 45]. The most prominent works, however, are mainly based on the textual content of the Web pages. This approach, although it proves effective in the generation of reasonable sitemaps, turns out 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 with little textual information, such as pages rich in 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 a 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. For example, let us consider a website of an university department. Sitemaps generated on the basis of the textual content can 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 considered as siblings in the hierarchy by Web masters are spread into several parts of the extracted hierarchy (i.e. different research areas).

This paper is a contribution in the direction of extracting sitemaps automatically by combining information on the Web page structure and hyperlink structure of websites. This goal is achieved by analyzing Web page 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 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 based on the "distributional hypothesis", initially defined for words in natural language processing (i.e. *"You shall know a word by the company it keeps"*) [8] and recently extended to generic objects [15]. In our context, we transpose this idea in *"You shall know a Web page by the paths it keeps"*. According to this hypothesis, two Web pages are siblings in the hierarchy if they share the same most frequent path from the homepage in the website graph. In order to encode and identify "frequent paths from the homepage in the website graph", we resort to the data mining task of sequential pattern mining, which is a data mining task specialized for analyzing sequences of items, to discover sequential patterns. More precisely, it consists of efficiently discovering frequent subsequences in a set of sequences where, in our case, items represent Web pages and sequences represent paths in the website

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

graph. Additionally, in order to reduce time-complexity problems of the task of sequential pattern mining and reduce the presence of redundant patterns, the approach we follow is that of extracting *closed* sequential patterns, instead of extracting classical frequent sequential patterns. An additional reason for this choice is that in closed sequential pattern mining only closed patterns are discovered, allowing us to prefer shorter paths to longer paths when longer paths do not add much information. The immediate effect is to prefer the generation of shallow sitemaps to deep sitemaps.

To better explain this aspect, let us consider a website where a is the homepage, b is a webpage reachable from a, and c is a webpage reachable from b. If we consider that, by definition, a sequential pattern α is closed if it has no proper supersequence β with the same support, the idea is to prefer the sequence $\alpha = \langle a, b \rangle$ (or the sequence $\gamma = \langle a, c \rangle$) to the sequence $\beta = \langle a, b, c \rangle$, if α (or γ) has the same support of β. In fact a long path in the sitemap is (i.e., deep sitemaps are) only necessary if it (they) really provide additional information, but in this case all the paths that reach b from a, also reach c (after b).

The algorithm we propose, named *SMAP* (SiteMAP miner), implements a four-step strategy that: 1. generates the Web graph using Web-lists, 2. extracts sequences which represent navigation paths by exploiting the random walk theory, 3. mines frequent closed sequential patterns of navigation paths and 4. transforms discovered patterns in order to extract the sitemap in the form of a hierarchy. As previously mentioned, contrary to other approaches for sitemap generation that use the hyperlink structure, SMAP uses also the Web page structure (i.e. Web lists), which allows us to concentrate on a subset of links which describe the 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 presents an extended and revised version of the workshop paper presented in [23], which is a proof-of-concept for the four-step strategy implemented in SMAP. However, every single step has been revised from a methodological viewpoint. The main changes are: *i)* the extraction of navigation paths has been modified by considering not only *normal closed* sequential patterns, but also *contiguous closed* sequential patterns [30]. *ii)* the pruning strategy has been revised and integrated in the mining step, while in [23] it was implemented as a post-processing step. These changes have led to different results, so, in this paper, we have reported a comprehensive and more detailed empirical evaluation, also adding a complete comparison with state-of-the-art methods (not present in [23]). Finally, related works have been discussed in depth and analyzed.

The rest of the paper is organized as follows. In Section 2 we briefly discuss existing related work. In Section 3, we formally define the problem and introduce the necessary definitions. In Section 4 we introduce and describe our method for sitemap extraction. In Section 5 we describe the experimental setting, present the obtained results and comment on them. Finally, in Section 6 we draw some conclusions and outline possible future work.

2 Related work

To the best of our knowledge there is no work in the literature that uses sequential pattern mining for the automated extraction of sitemaps. This work, however, was motivated by 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 fields.

2.1 Sitemap extraction

The problem of generating website hierarchies is part of the more general research topic of Web mining. Although extracting website sitemaps is an important task for many Web applications, few studies specifically focus on this problem. In the following we revise some related works from the Web mining research area, even though they are not directly related to the specific task of sitemap generation. We classify them on the basis of the input data they use.

The *first category* includes Web Content Mining algorithms 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 [1] 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 to disambiguate words and construct the proper taxonomy. This is especially true for Web pages which have little textual information or Web pages written in a collaborative manner which therefore have different distribution of terms. Differently from traditional textual documents, Web pages are characterized by structural features, such as hyperlinks or an HTML structure which provide different, complementary information.

The *second category* includes Web Structure Mining algorithms 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 of Web structure mining algorithms is that if there is a hyperlink between two pages, then some semantic relation may exist between them [5, 22, 27]. 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 into a conceptual level of the sitemap 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 to the presence of short-cut links, which connect Web pages belonging to deeper levels of the hierarchy at shallow levels.

A more sophisticated approach is presented in [27], 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 with high frequency and hub value which describe the sitemap. In this case, the accuracy of the method strongly depends on the task of block extraction. In fact, the algorithm requires blocks to have the same structure on the Web pages. This assumption holds for blocks which are part of the navigation system in the Web pages which are firstly accessed by the user and typically represent the main content of the Web site. However, it is not true for more specific pages and pages at deeper levels of the website. In fact, many websites change the layout of secondary menus in deeper Web pages, to provide users with a sense of progression through the website. For this type of website, the proposed algorithm may fail to properly identify nodes of the sitemaps which correspond to pages at deeper levels of the Web site. Another drawback is due to the possible presence of false positive blocks that are included because they are recurrent, albeit not belonging to the navigation system (e.g., advertisements).

In [19] the authors focus on the problem of extracting structured data such as menus from Web pages, in order to identify the main hierarchical structure and the boundaries of a website. This is done by analyzing cliques among pages in the Web graph. As in [27], the algorithm segments Web pages of a given website into blocks and identifies the blocks that compose the maximal cliques as the main menus of the website. Although the method is unsupervised and thus suitable for the analysis of heterogeneous websites, it may suffer from the same limitations as [27] when processing deeper levels of the website.

Other state-of-the-art algorithms combine the hyperlink structure with content information of Web pages to extract the hierarchical organization of a website. In [45] 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 the results seem to be accurate enough, labeling examples requires a lot of effort. Therefore, the solution is not directly applicable in huge, heterogeneous websites. In the same line of research, [40] proposes a method (HDTM) that extracts document-topic hierarchies from websites. This method combines information from text and hyperlinks by alternating two steps: 1) Random walk with restart from the homepage with the assignment of an initial multinomial distribution to each node (i.e., page); 2) Update of the multinomial distribution when the walker reaches a node. These steps are performed using the Gibbs sampling algorithm and several thousand Gibbs-iterations are usually required before a hierarchy emerges. The resulting hierarchy is obtained by 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. Although this method is similar to ours in its purpose, it does not exploit web-lists and constructs document-topic hierarchies in a buttom-up fashion. In the experiments we will compare our method and HDTM.

2.2 Sequential pattern mining for Web mining

Our work is also connected with some works in the field of Web Usage Mining, when the goal is to apply sequential pattern mining algorithms for identifying patterns in Web log files. Such patterns describe how users navigate in the website [4, 30]. In general, Web usage mining algorithms, which extract hierarchies based on user behavior analyze two different types of patterns: *sequential patterns* and *contiguous sequential patterns* [30]. 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 [32] the authors compare models based on sequential patterns and contiguous sequential patterns for predicting the next page that the user will reach. They claim that models based on contiguous sequences perform better on websites with a 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, to 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., the hierarchies depend on the user profile) and only consider Web pages having a user-specified number of visitors.

Independently of the related work that use sequential pattern mining in Web mining, one of the main topics covered by this paper is that of mining closed sequential patterns. If compared with the more common problem of sequential pattern mining, closed sequential pattern mining approaches avoid the generation of unnecessary sub-sequences, thus leading to more compact results and saving computational time and space costs [9, 11]. In the literature, there are many methods for mining closed sequential patters, the most prominent ones are: CloSpan [44], BIDE [38], ClaSP [14] and COBRA [17]. However, all these methods follow the same enumeration strategy: patterns are generated on the basis of the lexicographic ordering and this ordering is then used both in item extension and in sequence extension. However, in general, this pattern-growth strategy may present two drawbacks: redundant itemset extension and expensive "matching cost" in the generation of projected databases.

For this reason, we adapt [11] in SMAP, in order to extract a sitemap in terms of closed sequences of Web pages rooted in the homepage, by taking as input the website hyperlink graph structure, as well as the structural and visual information of Web pages.

Recently, high-utility pattern mining [10, 12] has been used as an alternative to closed sequential pattern mining to reduce number of the extracted patterns by focusing on implicit or potential information. The main advantage of high-utility sequential patterns is that they give the opportunity to leverage factors such as the utility, interest, risk, and profit of items [13]. Although these factors can also be important for the task at hand, it would be necessary to define a-priori some factors for the items we consider (Web pages) and for the sequences they generate. This process, however, requires domain knowledge and can be subjective. Instead, in this paper, we follow a frequency-based approach to extract closed sequential patterns of interest and we postpone the application of high-utility sequential patterns for future research.

2.3 Automatic extraction of Web lists

As claimed in [5], not all the links are equally important to describe the website structure. Several works in the field of Web mining exploit Web pages taking advantage of the structural and visual information embedded in the HTML tags. In [5, 22, 26] collections of hyperlinks having similar visual and/or structural properties are used to filter noisy links and collect Web pages belonging to same semantic type. In [5, 26] the aim is to exploit Web lists for the task of Web page clustering. Specifically, in [5] 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. Moreover, in [26] 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 [22] 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 and 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: useful definitions

Before describing the methodology we propose for the extraction of sitemaps, we provide some important definitions. Such definitions characterize the structure of Web pages and the hyperlink structure of Web sites.

A Web page is characterized by multiple modalities. They can be associated to what they represent: a textual representation (i.e.., textual content), a visual representation (i.e., Web page rendering) and a structural representation (i.e., HTML organization). The method we propose takes into account both the visual and the structural representations. Therefore, in the following, we provide definitions which formally identify both representations.

Definition 1 A Web page is characterized by its **Structural Representation** which consists of Web elements present in the HTML code and organized according to a tree structure. HTML tags refer to text, hyperlinks and multimedia data, to give them different meaning and rendering in the Web page.

Definition 2 Web Page Visual representation. Let us consider a Web page rendered by a Web browser according to the CSS2 visual formatting model [25], which represents the Web page elements as rectangular boxes characterized by their position and geometry. Then, by taking into account these properties and the spatial relationships they define (e.g., *next_to*, *inside*) it is possible to create a tree, called **Rendered Box Tree**. In a 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 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 [25]. Moreover, the Rendered Box Tree could have a completely different structure from that of the corresponding HTML tag tree. This is because *i)* a Web page can be enriched with 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.

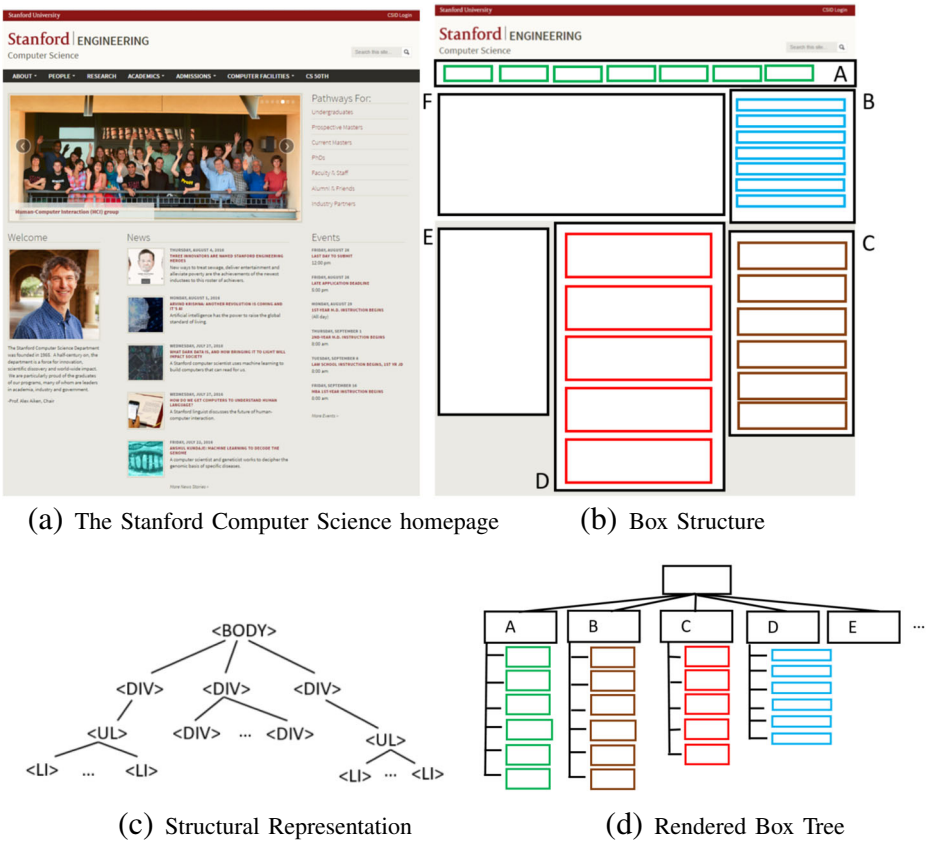
Example 1 Figure 1c and d show an example of the structural and visual representations for the homepage of a Computer Science Department website.

Leaves of the structural tree and leaves of 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 that shown in Figure 1d (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 structure, are used in the formal definition of Web lists, which is crucial for our approach:

Definition 3 A **Web List** is a set of several Web elements (called data records), such that:

i) Their rendered boxes have a similar HTML structure.



(a) The Stanford Computer Science homepage (b) Box Structure

(c) Structural Representation (d) Rendered Box Tree

Figure 1 The Stanford Computer Science homepage: Web page (**a**), Box Structure (**b**), Structural representation (**c**) and Rendered Box Tree (**d**). Each Web list is represented with a different color in (**b**) and (**d**)

ii) They are 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) [22].

This definition requires an algorithm which checks whether the 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 1b we show an example of a Computer Science Department website. From the illustrated page, one horizontal Web list (box A) and three vertical Web lists (boxes B, C, D) can be extracted.

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

Definition 4 Let V be the set of Web pages of a Web site, $h \in V$ be the homepage and E be the set of directed hyperlinks between two Web pages, then a **Website** is formally defined as a directed graph $G = (V, E, h)$.

The homepage h of a website is an important component of a website since it represents the entry point and allows the website to be seen as a rooted directed graph.

Example 2 Figure 2a show an example of a Web graph, where nodes are Web pages and edges are hyperlinks.

As previously mentioned, the algorithm we propose exploits a sequential pattern mining step. The basic idea is to consider a navigation path in a website as a sequence of urls (links) and use sequences of such urls to identify frequent navigation paths. Formally, a sequence is defined as follows:

Definition 5 Sequence: Let $G = (V, E, h)$ be a website, then 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 j-th Web page found in the navigation path S.

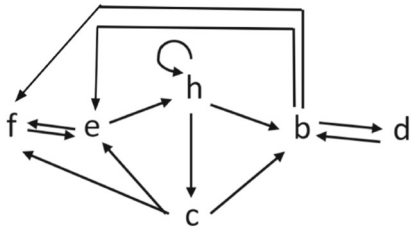
A sequence $S = \langle t_1, t_2, \ldots, t_m \rangle$ is a **sub-sequence** of a sequence $S' = \langle a_1, a_2, \ldots, a_n \rangle$, if and only if integers $i_1, i_2, \ldots, i_m$ exist, such that $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ and $t_1 = a_{i1}, t_2 = a_{i2}, \ldots t_m = a_{im}$. Moreover,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

Example 3 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 e is not equal to any Web page 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* $\sigma(S)$ (or, simply "support") is the absolute support divided by the size of the database (i.e., $|SDB|$). Formally:

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

A sequence is frequent when its support is higher than a user-defined threshold. In our work the support defines the relationship strength among the Web pages.



(a)

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>

(b)

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

(c)

Figure 2 **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 walk with restart from the homepage; **c** VIL for the sequence $\alpha = \langle h, b \rangle$

Example 4 Figure 2b shows an example of a database of sequences SDB. In this database the sequence $S = \langle h, b \rangle$ has relative support $\sigma(S) = 0.83$.

Following the suggestion provided in [30], 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).

Example 5 Given the Web graph of Figure 2a, a *contiguous* sequence is $\langle h, b, d \rangle$, while a *normal* sequence is $\langle h, d \rangle$.

The task of sequential pattern mining corresponds to the task of extracting frequent (sub-)sequences (non-contiguous or contiguous) from SDB. This is considered challenging from a time complexity view point because algorithms have to generate and test a combinatorially explosive number of intermediate sub-sequences. Moreover, in sequential pattern mining, most of the extracted sequences are very similar to others and highly redundant. For these two reasons, we concentrate on the problem of extracting *closed* sequential patterns, instead of extracting frequent sequential patterns. This task is less computationally demanding and reduces the problem of redundancy in the extracted patterns.

A closed sequential pattern (or simply a closed sequence) is defined as follows:

Definition 7 Closed Sequence: A sequential pattern S_j (either contiguous or not) is **closed** if and only if it is frequent and its support is different from that of all its frequent super-sequences.

Example 6 Figure 2b 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, sequence $\langle h, d \rangle$ is not closed because the super-sequence $\langle h, b, d \rangle$ has the same support as $\langle h, d \rangle$.

The last definition we have to provide before discussing the technical details of the method is that of the sitemap. This definition is particularly important since it implicitly 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 a *Web list* of the Web page j_1 (See Definition 3);
4. $\forall e \in E_i',\ w(e) \geq t$.

The meaning of $w(\cdot)$ and the way the tree T is generated remain unspecified in this (general) definition. These aspects are intentionally kept as parameters in Definition 8, in order to emphasize the general applicability of the framework. We only anticipate that our solution does not need to generate all the possible trees $T_i \subseteq G$ to optimize $\Phi(\cdot)$, but it is able to extract T directly, by analyzing a sub-graph of the website.

4 Methodology

The problem we consider is extracting the website's sitemap, according to Definition 8. To achieve this goal, we combine different techniques which have their roots in Web mining and data mining research areas: *i)* Information extraction for identifying potential navigation systems in Web pages in the form of web-lists; *ii)* The 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 structural and visual Web page information to extract Web lists in order to mitigate problems deriving from useless, noisy links. The output of this phase is the website graph G, where each node represents a single page and the edges represent the hyperlinks. In the second phase we generate sequences of urls (i.e., Web pages) which describe navigation paths. This is done by exploiting random walks extracted from the crawled website graph. In the third phase we mine closed frequent sequences of urls (i.e., closed frequent navigation paths) in the form of a tree. Unfortunately, this step does not consist in the simple application of any (closed) sequential pattern mining algorithm, since it is necessary to generate a tree of Web pages (which represents the sequences) with specific characteristics: the root is the homepage, any Web page appears only once and it is not possible to have multiple paths that connect the homepage with a Web page.

4.1 Website crawling

As claimed in [5], not all links are equally important to describe the website structure. In fact, a website is rich in useless, noisy links, which may not be relevant for the sitemap extraction process. This is the case of hyperlinks used to enforce the Web page authority in a link-based ranking scenario, short-cut hyperlinks, etc. The solution we propose, based on the usage of Web lists, has a twofold effect: on the one hand, it guarantees that only hyperlinks which may belong to potential navigation systems are considered; on the other hand, it allows the method to identify hyperlinks by implicitly taking into account the Web page structure codified in the Web lists available in the Web pages [5, 22, 37, 42], even if the hyperlinks do not belong to the navigation system.

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 as h and organized in *Web lists*. Only urls (i.e. Web pages) included in Web lists are further explored (line 6). The output of the website crawling step is the graph $G = (V, E)$ which will be used in the next step. To avoid a Web page being crawled multiple times, a frontier stores the set of Web pages already analyzed.

The method *extractWebLists()* extracts Web lists according to Definition 3, where, however, some concepts have to be instantiated. The first concept is related to the analysis of the HTML structure. Specifically, to check whether two Web elements e_i, e_j have a similar HTML structure, we first convert each HTML tag into a unique character. Then, we codify the converted HTML tree, having as its root the Web element e_i (e_j), into a string composed of the converted HTML tags, ordered by applying a breadth search to the rooted tree. Finally, we apply to the generated strings the *Normalized Edit Distance*, which is the cost associated with the minimum set of operations (in terms of inserting, deleting and updating), needed to transform one string into another string [47].

Algorithm 1 crawlingWebsite(homepage)

Input: URL homepage;
Output: Set<(URL, URL)> E; Set<URL> V;

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

Other concepts to be instantiated in *extractWebLists()* are adjacency and alignment. Two Web elements e_i, e_j are adjacent if they are siblings in the Rendered Box Tree and there is no Web element e_w, different from e_i and e_j, which is rendered between them. Finally, two Web elements e_i, e_j are aligned on: *i)* the x-axis if they have the 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 when $e_i.x = e_j.x$ and $e_i.y = e_j.y$.

Algorithm 2 describes in detail the Web list extraction process: the algorithm analyzes all the nodes which, in the Rendered Box Tree, are children of the rectangular box representing the *body tag* element. For each node to be analyzed (line 5), the algorithm tries to align its children (lines 9-30), only if the number of children is relatively small. Otherwise, it only enqueues the children for further processing. The rationale is that only small subtrees of the Rendered Box Tree can represent weblists, whereas subtrees, rooted at the higher levels of the Rendered Box Tree, typically do not represent any structured data. In our experiments, the threshold value used for the size of the tree (*maxSize*) is 30. This value guarantees that navigation systems are considered for the alignment. Only the nodes which are included neither in *verticalLists* nor in *horizontalLists* are further explored by the algorithm (lines 27-28). The method *getStructurallySimilar()* checks whether the elements in a list have a similar HTML structure.

4.2 Sequence database generation

To capture and codify correlations among graph 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 by setting a single node (i.e. the homepage) as the starting point. 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 [46].

Algorithm 2 extractWebLists(webpage)

```
Input: URL webpage;
Output: List<List<URL>> W; //list of weblists.
body = webpage. findElementByTagName("body");
notAligned = body. getChildren();
repeat
    node = notAligned. dequeue();
    children = node. getChildren();
    if node.getSize() >= maxSize then
        notAligned. enqueue(children);
    else
        verticalAligned= children.groupByX();
        horizontalAligned = children.groupByY();
        lists = List();
        for each L ∈ verticalAligned do
            structurallySimilar = getStructurallySimilar(L);
            if structurallySimilar.getSize() ≥ 2 then
                lists.add(structurallySimilar);
            end if
        end for
        for each L ∈ horizontalAligned do
            structurallySimilar = getStructurallySimilar(L);
            if structurallySimilar.getSize() ≥ 2 then
                lists.add(structurallySimilar);
            end if
        end for
        for each L ∈ lists do
            W.add(L. getUrls() )
        end for
        notIncluded = children - lists.getWebElements();
        notAligned. enqueue(notIncluded);
    end if
until !notAligned.empty() return W
```

Using this approach it is more likely that the Web pages closer to the homepage will 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. In addition, with this approach we also guarantee the likelihood that Web pages that are easily accessible from the home page will be reached through shortcuts or multiple paths.

Formally, given the Web graph G extracted by means of the website crawling step and two numbers $rwrLength, dbLength \in \mathbb{N}$, random walks are used to navigate the Web graph G from the homepage h (which is one of the nodes of G). The output of this step is a database of sequences SDB composed of $dbLength$ random walks with maximum length $rwrLength$ and starting from h.

As observed in [36], if we increase the length $rwrLength$ of a random walk starting at node i, the probability of reaching a node j tends to depend on the degree of j. The effect

of this property is that with long random walks it is easy to reach "hubs", which might mean moving from one community to another (i.e., from one part of the Web site to another, in our case). The same authors also observed that, on the contrary, if we reduce the length $rwrLength$, the random walks tend to become "trapped" in densely connected parts of the graph corresponding to communities, without necessarily reaching hubs. In our approach we exploit this property and generate short random walks so that the random walks become "trapped" in densely connected pages of the website. The aim is to avoid inferring false correlations among Web pages due to the nodes' degree.

Algorithm 3 describes the generation process of the sequence database or, equivalently, the set of navigation paths. Figure 2b shows a sequence database obtained from the Web graph in Figure 2a. The generation process is exemplified in the following example:

Example 7 Let us consider the Web graph in Figure 2a. Here, each node is a Web page and there exists an edge between two nodes t_i and t_j if and only if the Web page t_i has an outlink to the Web page t_j. Therefore, starting from the homepage h, a random walker can reach in one hop the nodes (i.e. Web pages) h, b, c or e. Suppose it randomly chooses the node b, then the *partial* sequence $\langle h, b\rangle$ is generated at the first step. At the second step, starting from the node b, the random walker can reach the nodes d, e or f. Suppose the node f is chosen, generating the partial sequence $\langle h, b, f\rangle$ and then, according to the same procedure, generating the sequence $\langle h, b, f, e\rangle$. The random walker can then decide to restart, making the sequence $\langle h, b, f, e\rangle$ a "final" sequence to be added to the sequence database. After restart, the random walker can generate, for instance, the second sequence, that is, the sequence $\langle h, b, d, b\rangle$ in Figure 2b. The process can then iterate and generate all the sequences in the figure.

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 λ > α 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 forreturn randomWalks
```

4.3 Mining sitemaps

Given the sequence database (SDB), extracted according to the procedure described in Section 4.2, and a user-defined threshold t (i.e. minimum support), we exploit a closed sequential pattern mining algorithm to extract all the frequent sequences to be used for

sitemap generation. In this phase we use a revised version of CloFAST [11] as the closed sequential pattern mining algorithm. We chose CloFAST because it provides compact results, while saving computational time and space, if compared with other closed sequential pattern mining algorithms.

Another reason for using CloFAST is that it is able to extract *closed* sequential patterns. In our context, if a path $h \rightarrow c \rightarrow f$ has the same support as $h \rightarrow c \rightarrow f \rightarrow e$, probably e should be included in the sitemap because it has, at least, the same importance as f. On the other hand, ignoring the sequence $h \rightarrow c \rightarrow f$ does not affect sitemap construction since $h \rightarrow c \rightarrow f \rightarrow e$ is still kept. Moreover, mining closed sequential patterns is a less demanding task than mining (frequent) sequential patters, since mining closed sequential patterns can lead to a reduction of the search space.

In its original version CloFAST returns a tree $T_{CloFAST} = (V', E')$ and a weight function $\sigma : E' \rightarrow \mathbb{R}$. In $T_{CloFAST}$ each node $v \in V'$ has a label and σ associates each node $v \in V'$ with the *relative* support of the sequence $\langle t_1, t_2, \ldots, t_m \rangle$ such that t_i is a label associated to the i-th node found in the path between the root and v (t_1 is the label associated to the root of $T_{CloFAST}$ and t_m is the label associated to the node v).

In our modified version of CloFAST labels represent urls. Moreover, the way CloFAST generates sequential patterns has been modified, in order to take into account additional constraints.

- The first constraint does not allow the generation of frequent sequences which start from a node that is not the homepage;
- The second constraint (optional) ensures that frequent paths, that do not exist in the crawled graph G, are removed (see Definition 6, Contiguous sequence.);
- The third constraint prevents two nodes in the tree $T_{CloFAST}$ being labeled with the same label.

The reason for the first constraint is quite intuitive (sitemaps start from the homepage), while the second and third constraints require additional motivations. The second constraint is necessary in order to compare the sitemaps extracted by considering sequential patterns with those extracted by considering contiguous sequential patterns. In its original version, CloFAST is not able to extract contiguous sequential patterns. Concerning the third constraint, we have to note that the tree $T_{CloFAST}$, extracted with the application of the original version of CloFAST, may contain multiple frequent paths that reach, from the homepage, a given Web page (node label). For example, in Figure 2a the Web page e can be reached using the paths $h \rightarrow c \rightarrow e$ and $h \rightarrow c \rightarrow f \rightarrow e$, and both can be frequent. However, in the sitemap each Web page can be reached by only one path starting from the homepage.

Example 8 Figure 3a and c show trees extracted by CloFAST, where constraints 1,3 and 1,2,3 are satisfied, respectively. In both cases the trees have been extracted from the SDB in Figure 2b.

Before discussing the way constraint checking is implemented, we have to describe one of the main data structures that CloFAST uses: the *Vertical id-List* (VIL). VILs are used by CloFAST for support counting 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

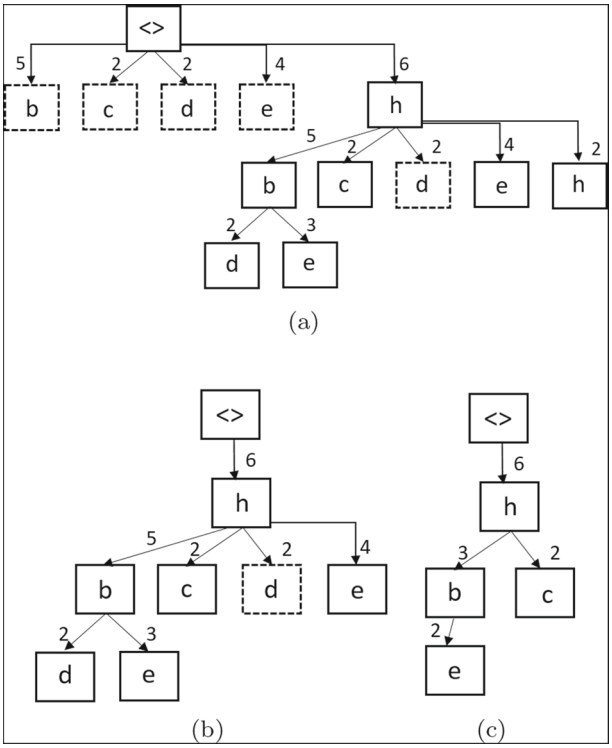


Figure 3 **a** A frequent sequence tree extracted by CloFAST with absolute minsup = 3 from the SDB in Figure 2b. Nodes with dashed borders represent non-closed nodes; **b** A frequent sequence tree extracted by the extended version of CloFAST, having the support as its weight function; **c** A contiguous sequence tree extracted by the extended version of CloFAST, having the contiguous support as its weight function

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.

Example 9 Figure 2c shows the VIL of sequence $\alpha = \langle h, b \rangle$. The values in VIL_α represent the end position of the occurrences of sequence α in the sequences of Figure 2b. For instance, the first element (list "[2]") represents the position of the first occurrence of Web page b, after Web page h, in the first sequence ($SID = 1$). The second element (list "[2,4]") represents the positions of b after h in the sequence S_2. The other values are respectively the list with only value 3 (for sequence S_3), the list with only value 2 (for S_4), $NULL$ for S_5 and the list with only value 3 (for S_6)

The checking of the first constraint is implemented, in the new version of CloFAST, by working on VILs. Specifically, the VILs are constructed level-wise (for sequences of increasing length). In order to avoid the generation of patterns, whose first element is different from the homepage, firstly we generate one single VIL, that is, the one associated

with the sequence $\langle h \rangle$. Other VILs (for other sequences of length 1) are simply not generated. This sequence is the only one which is further extended in the subsequent steps of the algorithm.

VILs are also used to check the second constraint. In particular, starting from the root of $T_{CloFAST}$, the function *contiguous(·)* (see Algorithm 4) is iteratively applied to update the node VILs. This function looks for possible "holes" in the sequences by bottom-up climbing the sequence tree $T_{CloFAST}$. A hole is found when the condition at line 10 is not satisfied. The returned VIL of a node u is used to calculate its (absolute) *contiguous* support, without scanning the sequence database SDB: $\sigma_c(u) = |\{j | vil = contiguous(u, T_{CloFAST}) \wedge vil[j][1] = h\}|$. Therefore, if the contiguous support of u, $\sigma_c(u)$ is greater than the threshold t, the node is kept, otherwise it is pruned. Note that pruning is possible because of the non-monotonic behaviour of the contiguous support with respect to the length of the sequences. Moreover, this approach does not require sequence database scanning.

As for the third constraint, since in the sitemap each Web page can be reached by only one path starting from the homepage, we have to select, for each Web page included in the frequent sequence tree (or contiguous sequence tree), the best path to reach it. Formally, a node $u \in T_{CloFAST}$ ($u \in T'_{CloFAST}$ for contiguous sequences) is pruned if $\exists v \in T_{CloFAST}$ ($\exists v \in T'_{CloFAST}$ for contiguous sequences), such that $u \neq v$, $l(u) = l(v)$ and $\sigma(u) < \sigma(v)$ ($\sigma_c(u) < \sigma_c(v)$ for contiguous sequences), with $l(u)$ being the label associated with node u.

In this way, we implicitly instantiate the function $w(a)$, introduced in Definition 8, as the maximum support (or contiguous support) of sequences in $T_{CloFAST}$ which represent navigation paths from the homepage to a.

4.4 Time and space complexity

The time complexity of the website crawling stage is $O(|G| * num\_lists * num\_urls)$, where $|G|$ is the number of Web pages belonging to a website, num_lists is the number of extracted weblists for each Web page, and num_urls is the number of urls in a Web list. As for the sequence database generation phase, we have $O(rwrLength * dbLength)$, where $rwrLength$ is the length of a random walk and $dbLength$ is the number of random walks.

However, the main source of complexity comes from the sequential pattern mining task, which is known to be exponential in the depth of the trees [31] which, in our case is bounded by rwrLength (which is a relatively small number). More specifically it is, in the worst case, linear in the number of nodes of the frequent sequence tree and linear in dbLength, where the number of nodes of the frequent sequence tree can be exponential in the depth of the trees. However, we have to clarify that the time complexity of sequential pattern mining algorithms depends on the number of patterns in the search space (and thus, on the value of the minimum support threshold), and the cost of the operations for generating and processing each itemset. In this respect, CloFAST implements an efficient algorithm to prune the search space in the extraction of closed sequential patterns, which is $O(d \cdot dbLength)$, where d is the depth of the tree [11]. Finally, the algorithm used to extract contiguous patterns only works on VILs, already generated by CloFAST to identify closed patterns. Therefore, searching of contiguous patterns, does not requires additional effort. This makes the added time complexity the same as the pruning in CloFAST, that is, $O(d \cdot dbLength)$, with the advantage of further reducing the search space.

As for the space complexity, the required space to store the sequence database is $O(rwrLength * dbLength)$. Moreover, the space required to run the sequential pattern mining task is due to the size of the trees and the size and number of the VILs. This last aspect is prevalent and makes the space complexity linear in the number of nodes of the

Algorithm 4 contiguous(T,u)

Input: T: a sequence tree extracted by CloFAST; u: node of T
Output: vil: the VIL of the sequence at node u such that the contiguous condition is satisfied.

```
1:  vil = getVil(u);
2:  parent = getParent(u);
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:     u = parent
21:     parent = getParent(u);
22: until parent != root(T) return vil;
```

frequent sequence tree and linear in $dbLength$. The main problem is that the number of nodes of the frequent sequence tree can be exponential in the depth of the trees but, as already pointed out, in our case the depth of the trees is bounded by $rwrLength$.

5 Experiments and discussion

The method described in this paper has been implemented in the system SMAP and, in this section, we present the results of its empirical evaluation. We first investigate the influence of the threshold t. Next, we focus on the influence of the number and length of random walks. Then, we compare the contribution of using (non-contiguous) closed sequential patterns with the contribution of using contiguous closed sequential patterns. Finally, we compare SMAP with some state-of-the-art methods and one baseline method.

5.1 Datasets

The evaluation of SMAP is mainly based on the quality of sitemaps extracted. This, however, is not trivial task because, to the best of our knowledge, there is no dataset publicly available for the specific task of sitemap extraction. Therefore, we considered two possible solutions: *i)* involving human experts to extract the hierarchical organization of websites

and generating a ground-truth dataset for each considered website; *ii)* using, as ground-truth, existing and real sitemap pages, which are manually generated by Web masters and available for some websites.

Obviously, the first solution would have required extensive human effort and working time. This problem becomes more evident if we consider websites with a great number of strongly connected Web pages and websites with deep hierarchies.

In the second case, sitemap pages are in general manually created by Web designers. However, the risk is that sitemap pages are not updated (i.e. they can contain links to non-existent pages or they ignore the existence of new sections) or are very abstract (i.e. contain shallow hierarchies composed of few pages). For this reason, to empirically evaluate SMAP, we performed experiments on the following websites which provide updated sitemap pages: www.cs.illinois.edu, www.cs.ox.ac.uk, and www.cs.princeton.edu.

The main characteristics of the three datasets are reported in Table 1. As it can be seen, they vary significantly in the number of Web pages, degree of connection and degree of connection in Web lists.

5.2 Experimental setting

The evaluation was performed in terms of Precision, Recall and F-measure of edges. In particular, Precision measures how many of the extracted edges belong to the real sitemap. Recall measures how many edges, belonging to the real sitemap are extracted. We also include results in terms of F-Measure, the weighted harmonic mean of Precision and Recall. The results refer to the first two levels of the sitemaps, that is level 1 and level 2, while level 0 represents the root. Additional levels are hard to evaluate and, in most cases, do not exist in real sitemaps.

To investigate whether the observed differences in performance among the methods are statistically significant, we follow the recommendations outlined by Demšar [6]. Specifically, we use the non-parametric Wilcoxon paired signed rank test [43] to compare the predictive performance of two methods over multiple datasets. On the other hand, for comparison of multiple methods, we use the corrected Friedman test and the post-hoc Nemenyi test [33]. We present the result from the Nemenyi post-hoc test on an average rank diagram. The ranks are depicted on an axis, so that the best ranking algorithms are at the right-most side of the diagram. The algorithms that do not differ significantly (in performance) are connected with a line. In all the experiments reported in this study the significance level is set to 0.05.

The results are compared with those obtained by HDTM [40]. For HDTM, we set $\gamma = 0.25$ (to avoid hierarchies too shallow or too deep) and 5,000 Gibbs iterations, as suggested by the authors in their paper.

We also compare SMAP with an algorithm (called MST) which, starting from graph G (the same as that built in SMAP), generates as a sitemap a maximum spanning tree. The

Table 1 Datasets description

Website	No. of Web pages	No. of hyperlinks	No. of hyperlinks in Web lists	No. of edges in (real) sitemaps
www.cs.illinois.edu	563	9415	5330	54
www.cs.ox.ac.uk	3480	44526	35148	40
www.cs.princeton.edu	3132	122493	104585	44

maximum spanning tree is created, as for SMAP, by using random walks. In particular, graph G is used to generate a new weighted graph G', where nodes and edges are the same as G, whereas the weight associated to each edge represents the number of walks that cross that edge. G' is then used to generate sitemaps using any maximum spanning tree extraction algorithm (in our implementation we used the Kruskal's Algorithm [21]). Since SMAP and MST share many steps, the purpose of the comparison is to "isolate" and evaluate the contribution provided by the revised version of CloFAST in the identification of sitemaps.

5.3 Results: influence of parameters

The three main parameters (i.e., t: minimum support, $rwrLength$: size of random walks, $dbLength$: number of generated random walks) were compared by focusing on the following possible values: $t = \{0.0001, 0.0005, 0.001, 0.005\}$, $rwrLength = \{5, 7, 10\}$ and $dbLength = \{100K, 500K, 1M\}$. The experiments were run according to a "grid search", where each combination of parameter values were tested and the results are shown by aggregating them (in terms of average) on a single parameter.

Table 2 Evaluation of SMAP by varying the minimum (contiguous) support threshold

Website	t- minSupp	Precision @1	Recall@1	F@1	no. edges@1
www.cs.illinois.edu	0.0001	1	0.63	0.77	19
www.cs.illinois.edu	0.0005	1	0.63	0.77	19
www.cs.illinois.edu	0.001	1	0.63	0.77	19
www.cs.illinois.edu	0.005	1	0.63	0.77	19
www.cs.ox.ac.uk	0.0001	1	0.35	0.52	23
www.cs.ox.ac.uk	0.0005	1	0.35	0.52	23
www.cs.ox.ac.uk	0.001	1	0.35	0.52	23
www.cs.ox.ac.uk	0.005	1	0.35	0.52	23
www.cs.princeton.edu	0.0001	1	0.19	0.32	31
www.cs.princeton.edu	0.0005	1	0.19	0.32	31
www.cs.princeton.edu	0.001	1	0.19	0.32	31
www.cs.princeton.edu	0.005	1	0.19	0.32	31
Website	t- minSupp	Precision @2	Recall@2	F@2	no. edges@2
www.cs.illinois.edu	0.0001	0.90	0.31	0.46	123
www.cs.illinois.edu	0.0005	0.90	0.31	0.46	122
www.cs.illinois.edu	0.001	0.90	0.61	0.69	83.5
www.cs.illinois.edu	0.005	0.48	0.93	0.47	22
www.cs.ox.ac.uk	0.0001	0.59	0.17	0.26	112
www.cs.ox.ac.uk	0.0005	0.60	0.17	0.265	112
www.cs.ox.ac.uk	0.001	0.60	0.19	0.28	103
www.cs.ox.ac.uk	0.005	0.37	0.68	0.32	32
www.cs.princeton.edu	0.0001	0.88	0.17	0.28	198
www.cs.princeton.edu	0.0005	0.89	0.21	0.34	161
www.cs.princeton.edu	0.001	0.87	0.33	0.47	119
www.cs.princeton.edu	0.005	0.40	0.80	0.41	18

"@N" indicates the level number, where level 0 represents the root.

Table 3 Evaluation of SMAP by varying the parameter *dbLength* (see Section 4.2)

Website	*dbLength*	Precision @1	Recall @1	F @1	Precision @2	Recall @2	F @2
www.cs.illinois.edu	100K	1	0.63	0.77	0.80	0.54	0.52
www.cs.illinois.edu	500K	1	0.63	0.77	0.80	0.54	0.52
www.cs.illinois.edu	1M	1	0.63	0.77	0.8	0.547	0.52
www.cs.ox.ac.uk	100K	1	0.35	0.52	0.54	0.31	0.29
www.cs.ox.ac.uk	500K	1	0.34	0.51	0.53	0.30	0.28
www.cs.ox.ac.uk	1M	1	0.35	0.52	0.53	0.30	0.28
www.cs.princeton.edu	100K	1	0.19	0.32	0.73	0.36	0.37
www.cs.princeton.edu	500K	1	0.19	0.32	0.77	0.36	0.37
www.cs.princeton.edu	1M	1	0.19	0.32	0.79	0.42	0.39

"@N" indicates the level number, where level 0 represents the root.

Table 2 shows the effectiveness of SMAP by varying the minimum support. This table shows that when the minimum support threshold decreases we are able to obtain wider sitemaps. Consequently, as expected, by reducing the minimum support threshold precision increases and recall decreases. This is due to the fact that, by decreasing the support, the number of generated sequences increases and the extracted hierarchy becomes deeper and wider, including website sections which are not included in the sitemap page. By analyzing the F-measure we can see that $t = 0.001$ generally produces the best trade-off between precision and recall, especially at the second level of the hierarchy.

The results reported in Table 3 show that, in general, it is not necessary to generate a huge number of random walks: 100 thousand random walks for tens of thousands links is enough to obtain the best results. The situation is slightly different for www.cs.princeton.edu, where the different order of magnitude in the number of links requires more random walks. Additionally, in Table 4 we notice that longer random walks lead to higher precision and smaller recall, although this trend (even though it is intuitively motivated) is not statistically confirmed.

Finally, all the results show (as expected) that the first level of the sitemap is easier to identify than the second level. This is a natural consequence of the larger number of pages at

Table 4 Evaluation of SMAP by varying the *rwrLength* parameter (see Section 4.2)

Website	*rwrLength*	Precision @1	Recall @1	F @1	Precision @2	Recall @2	F @2
www.cs.illinois.edu	5	1	0.63	0.77	0.8	0.55	0.52
www.cs.illinois.edu	7	1	0.63	0.77	0.8	0.55	0.52
www.cs.illinois.edu	10	1	0.63	0.77	0.80	0.52	0.51
www.cs.ox.ac.uk	5	1	0.35	0.52	0.54	0.33	0.30
www.cs.ox.ac.uk	7	1	0.35	0.52	0.54	0.30	0.28
www.cs.ox.ac.uk	10	1	0.34	0.51	0.53	0.29	0.26
www.cs.princeton.edu	5	1	0.19	0.32	0.72	0.35	0.34
www.cs.princeton.edu	7	1	0.19	0.32	0.77	0.40	0.37
www.cs.princeton.edu	10	1	0.19	0.32	0.79	0.39	0.38

"@N" indicates the level number, where level 0 represents the root.

the second level. In this respect, we also performed experiments with the dataset Princeton, including the third level (so to obtain four-levels sitemaps). The best results obtained by SMAP for this task are: precision= 0.61, recall= 0.27, Fmeasure= 0.38, with the following configuration: contiguous, $rwrLength = 10$, $minSupp = 0.0001$, $dbLength = 100k$, and precision: 0.83, recall: 0.68, Fmeasure: 0.75, with the following configuration: normal, $rwrLength = 10$, $minSupp = 0.001$, $dbLength = 500k$. However, in this case, the sitemap is not able to provide a high-level and concise description of the website content, because the algorithm starts to produce a sitemap which is similar to the actual structure of the website.

5.4 Results: Contiguous vs. non-contiguous closed sequential patterns

The results in Table 5 show a comparison between contiguous and non-contiguous ("normal" in the table) closed sequential patterns. As can be seen, there is no difference at the first level, while at the second level contiguous sequences guarantee higher recall at the price of lower precision. This is to be expected, since the set of frequent closed sequential patterns includes the contiguous ones or, in other words, non-contiguous closed sequential patterns introduce additional constraints in the construction of sitemaps. This reflects on the size of the sitemaps: sitemaps extracted from normal closed sequential patterns are wider than those extracted from contiguous closed sequential patterns and, consequently, they are more difficult to browse and manage by the users.

5.5 Results: comparisons

In order to make comparisons, we focus on the results obtained by SMAP with $t = 0.001$ and contiguous closed sequential patterns. As we have seen before, the support does not signifi-

Table 5 Evaluation of SMAP by considering closed sequential patterns (normal) and contiguous closed sequential patterns (contiguous)

Website	Type of closed sequential patterns	Precision @1	Recall@1	F@1	no. edges @1
www.cs.illinois.edu	normal	1	0.63	0.77	19
www.cs.illinois.edu	contiguous	1	0.63	0.77	19
www.cs.ox.ac.uk	normal	1	0.35	0.52	23
www.cs.ox.ac.uk	contiguous	1	0.35	0.52	23
www.cs.princeton.edu	normal	1	0.19	0.32	31
www.cs.princeton.edu	contiguous	1	0.19	0.32	31
Website	Type of closed sequential patterns	Precision @2	Recall@2	F@2	no. edges@2
www.cs.illinois.edu	normal	0.95	0.48	0.50	103
www.cs.illinois.edu	contiguous	0.65	0.60	0.44	72
www.cs.ox.ac.uk	normal	0.66	0.23	0.33	99
www.cs.ox.ac.uk	contiguous	0.42	0.37	0.23	80
www.cs.princeton.edu	normal	0.88	0.35	0.44	150
www.cs.princeton.edu	contiguous	0.65	0.40	0.31	98

"@N" indicates the level number, where level 0 represents the root.

cantly influence the results, while the type of closed sequential patterns may lead to significant differences. We choose contiguous closed sequential patterns because, although they do not lead to the best F value, they produce smaller, more easily interpretable sitemaps.

In Table 6 we show the precision, recall and F measure of SMAP, MST and HDTM. Form the results, it is clear that, for sitemap extraction, the worst performing method is HDTM. This can be explained by the different nature of the two algorithms: contrary to SMAP and MST, HDTM organizes website pages in a hierarchy by using the distribution of Web page terms. Then, it could happen that, for example, for a Computer Science department website HDTM organizes 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 into different parts of the extracted hierarchy. At the third level, for the dataset Princeton, the HDTM results start to degenerate (precision: 0.09, recall: 0.02, F-measure: 0.03) for the same reasons described before, that is, because the sitemap is not able to provide a high-level and concise description of the website content.

The second observation we can make by analyzing the results reported in Table 6 is that there is no clear difference between SMAP and MST. While SMAP seems to perform better in terms of recall, MST seems to provide slightly better results than SMAP in terms of precision. On the contrary, in terms of F measure, SMAP always outperforms MST.

The statistical comparison of the three methods is reported in Figure 4. In this figure we can see that the best performing methods is SMAP, but there is no statistical evidence that SMAP outperforms MST in terms of precision.

To better analyze the pairwise comparison between SMAP and MST, in Table 7 we report the results of the paired Wilcoxon signed rank test. They show that SMAP is not superior to MST in terms of precision, but SMAP is clearly superior in terms of recall and F measure (especially at the second level of the sitemap). On the other hand, SMAP outperforms MST in terms of precision only at the first level of the hierarchy. This confirms that the usage of closed sequential patterns provides significant benefits in terms of quality of extracted sitemaps, if compared to simpler strategies like minimum spanning trees. The reason is that closed sequential patterns are based on probabilistic evidence, which is not the case of MST, which only exploits structural information.

Table 6 Precision, recall and F measure: We report the average (over the two levels) results of SMAP, MST and HDTM

		Precision	Recall	F	Number of edges in sitemaps
www.cs.illinois.edu	SMAP	0.929	**0.723**	**0.804**	107
	MST	**0.969**	0.460	0.625	217
	HDTM	0.285	0.310	0.290	32
www.cs.ox.ac.uk	SMAP	**0.767**	0.268	**0.397**	112
	MST	0.621	**0.297**	0.343	135
	HDTM	0.515	0.215	0.302	102
www.cs.princeton.edu	SMAP	0.919	**0.302**	**0.436**	155
	MST	**0.968**	0.179	0.323	230
	HDTM	0.640	0.125	0.210	223

SMAP results are obtained with t=0.001 and with contiguous closed sequential patters. The best result for the dataset/measure is reported in bold. The last column reports the number of edges in the extracted sitemaps

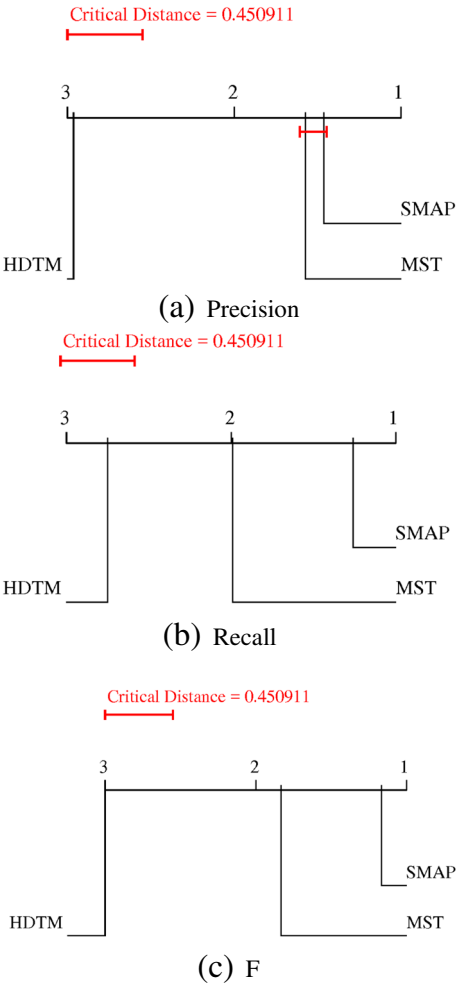


Figure 4 Nemenyi test for comparing multiple methods at different levels by considering all the datasets. The best configurations are positioned on the right. The algorithms that do not differ significantly are connected with a line

5.6 Qualitative results

A final analysis is aimed at qualitatively analyzing the sitemaps extracted. The first aspect we investigate is the size of the extracted sitemaps. From the last column of Table 6, we can see that SMAP extracts sitemaps which are quite stable in size. Actually, small changes in the size of the sitemaps reflect, somehow, the size of the dataset. On the contrary, by

Table 7 *p*-values of the Wilcoxon signed-rank test between SMAP and MST

	Precision	Recall	F
Lev 1	**0.001 (+)**	0.470 (+)	0.470 (+)
Lev 2	0.064 (-)	**0.001 (+)**	**0.001**(+)
Total	0.429(+)	**0.001**(+)	**0.001**(+)

In bold, we report significant *p*-values (< 0.05). '+' means that SMAP outperforms MST, whereas '−' means that MST outperforms SMAP

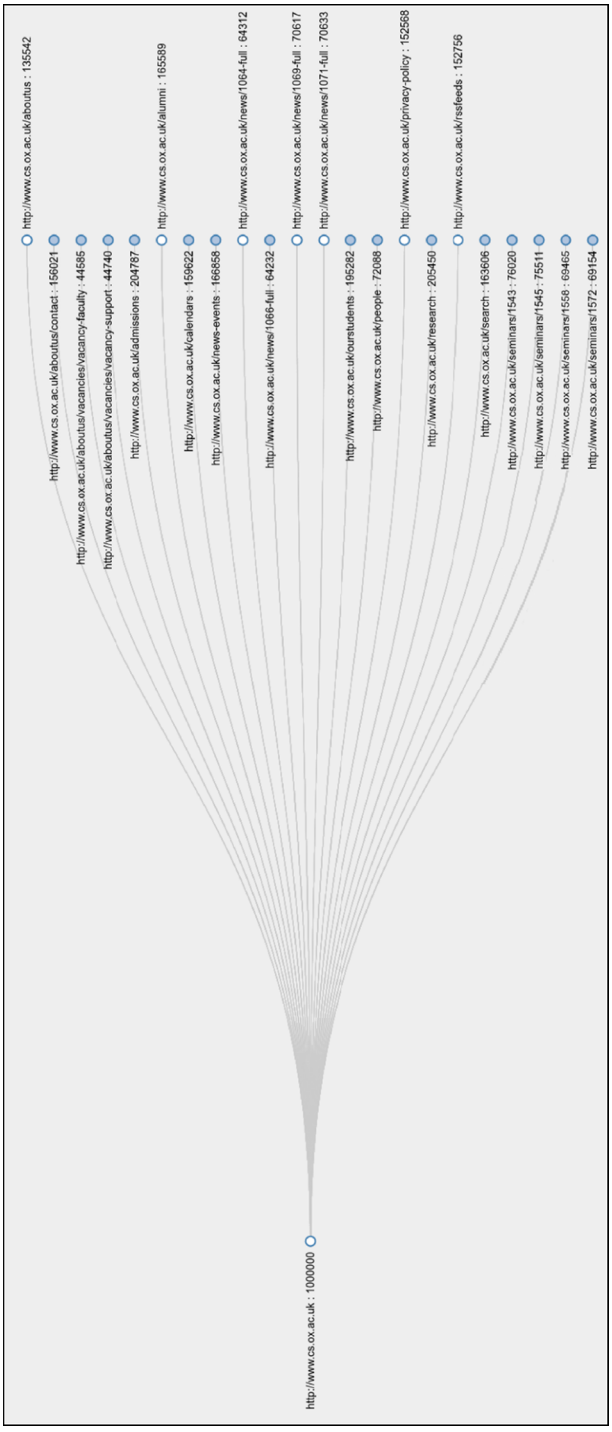


Figure 5 An example of a sitemap (only first level) extracted from www.cs.ox.ac.uk. SMAP was run with $t = 0.001$ and for mining contiguous closed sequential patternss

looking at the size of sitemaps extracted by HDTM, we see that there is high variability among the datasets. The effect we observe is that, for datasets with a large number of links, the generated sitemaps tend to overfit graph G (high precision, low recall), whereas if the number of links is relatively small, generated sitemaps tend to be quite general (recall > precision). As for MST, we can see that, although the size of sitemaps is quite stable, the extracted sitemaps contain many more Web pages, making them difficult to browse and view.

In Figure 5 we show an example of an extracted sitemap. All the Web pages identified by SMAP are present in the real sitemap, but only 35% of the Web pages present in the real sitemap are returned by SMAP. This is a clear indication that it is possible to simplify the real sitemap, since, many Web pages at the very first level are not easily reachable. This is due to the Web page structure and the hyperlink structure of the website. The solution is to make the Web pages more easily accessible, if necessary.

6 Conclusions

In this paper we have presented a new method for the automatic generation of sitemaps. SMAP successfully addresses the open issue of the extraction of hierarchical website organization, as understood and codified by Web masters through website navigation systems (e.g. menu, navbar, content list, etc.). Moreover, it provides Web masters with a tool for automatic sitemap generation, which can be helpful for explicitly specifying of the design concept and knowledge organization of websites.

The experimental results prove the effectiveness of SMAP compared to HDTM, a state-of-the-art algorithm, which generates sitemaps based on the distribution of Web page terms. The experiments also statistically prove the effectiveness of the sequential pattern mining approach, with respect to simpler and more common strategies, such as minimum spanning tree extraction.

As future work, we will investigate how to combine sitemaps, which describe the topical organization of a website (e.g. sitemaps extracted by HDTM), with sitemaps which describe the structural organization of the website. Moreover, we will investigate the usage of high-utility as an alternative to closed sequential patterns.

Acknowledgements We would like to acknowledge the support of the European Commission through the projects MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data (Grant Number ICT-2013-612944) and TOREADOR - Trustworthy Model-aware Analytics Data Platform (Grant Number H2020-688797). We would also like to thank Lynn Rudd for her help in reading and correcting the manuscript.

Funding Open access funding provided by Università degli Studi di Bari Aldo Moro within the CRUI-CARE Agreement.

Availability The source code and the datasets are available at the following hyperlink: https://github.com/fabiana001/sitemap-generator.

References

1. Aggarwal, C.C., Zhai, C.: A Survey of Text Clustering Algorithms. In: Aggarwal, C.C., Zhai, C. (eds.) Mining Text Data, pp. 77–128. Springer (2012)
2. Algosaibi, A.A., Melton, A.C.: Using the semantics inherent in sitemaps to learn ontologies. In: IEEE 38Th Annual Computer Software and Applications Conference, COMPSAC Workshops 2014, Vasteras, Sweden, July 21-25, 2014, pp. 360–365. IEEE Computer Society (2014)
3. Anderson, C.R., Domingos, P., Weld, D.S.: Adaptive Web navigation for wireless devices. In: Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'01, pp. 879–884. Morgan Kaufmann Publishers Inc., San Francisco (2001)
4. Baumgarten, M., Büchner, A.G., Anand, S.S., Mulvenna, M.D., Hughes, J.G.: User-driven navigation pattern discovery from internet data. In: Revised Papers from the International Workshop on Web Usage Analysis and User Profiling, WEBKDD '99, pp. 74–91. Springer, London (2000)
5. Crescenzi, V., Merialdo, P., Missier, P.: Clustering Web pages based on their structure. Data Knowl. Eng. **54**(3), 279–299 (2005)
6. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
7. Fang, X., Holsapple, C.W.: An empirical study of Web site navigation structures' impacts on Web site usability. Decision Support Systems **43**(2), 476–491 (2007). Emerging Issues in Collaborative Commerce
8. Firth, J.R.: Papers in linguistics 1934–51. Oxford University Press (1957)
9. Fournier-Viger, P., Lin, J.C.W., Kiran, R.U., Koh, Y.S.: A survey of sequential pattern mining. Data Science and Pattern Recognition **1**(1), 54–77 (2017)
10. Fournier-Viger, P., Lin, J.C.W., Nkambou, R., Vo, B., Tseng, V.S.: High-Utility Pattern Mining: Theory, Algorithms and Applications, 1St Edn, Springer Publishing Company, Incorporated (2019)
11. Fumarola, F., Lanotte, P.F., Ceci, M., Malerba, D.: Clofast: closed sequential pattern mining using sparse and vertical id-lists Knowledge and Information Systems (2016)
12. Gan, W., Lin, C., Fournier-Viger, P., Chao, H., Tseng, V., Yu, P.: A survey of utility-oriented pattern mining. IEEE Trans. Knowl. Data Eng., pp 1–1 (2019)
13. Gan, W., Lin, J.C., Zhang, J., Chao, H., Fujita, H., Yu, P.S.: Proum: high utility sequential pattern mining. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 767–773 (2019)
14. Gomariz, A., Campos, M., Marín, R., Goethals, B.: ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences. In: PAKDD (1), pp. 50–61 (2013)
15. Görnerup, O., Gillblad, D., Vasiloudis, T.: Knowing an object by the company it keeps: a domain-agnostic scheme for similarity discovery. In: Aggarwal, C., Zhou, Z., Tuzhilin, A., Xiong, H., Wu, X. (eds.) 2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015, pp. 121–130. IEEE Computer Society (2015)
16. He, D., Wu, D., Graves, W., Klein, M.: Creation of a DL by the Communities and for the Communities. In: 2019 ACM/IEEE joint conference on digital libraries (JCDL), pp. 327–328 (2019)
17. Huang, K.Y., Chang, C.H., Tung, J.H., Ho, C.T.: COBRA: closed sequential pattern mining using bi-phase reduction approach. In: Tjoa, A.M., Trujillo, J. (eds.) DaWaK, Lecture Notes in Computer Science, vol. 4081, pp. 280–291. Springer (2006)
18. Keller, M., Mühlschlegel, P., Hartenstein, H.: Search result presentation: Supporting post-search navigation by integration of taxonomy data. In: Proceedings of the 22nd International Conference on World Wide Web, WWW '13 Companion, pp. 1269-1274. ACM, New York (2013)
19. Keller, M., Nussbaumer, M.: Menuminer: Revealing the information architecture of large Web sites by analyzing maximal cliques. In: Proceedings of the 21st International Conference on World Wide Web, WWW '12 Companion, pp. 1025–1034. ACM, New York (2012)
20. Kim, D.J., Lee, S.C., Son, H.Y., Kim, S.W., Lee, J.B.: C-rank: A contribution-based Web page ranking approach. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, SAC '14, pp. 908–912. Association for Computing Machinery, New York (2014)
21. Kruskal, J.B.: On the shortest spanning subtree of a graph and the traveling salesman problem. In: Proceedings of the American Mathematical Society, pp. 7:48–50 (1956)
22. Lanotte, P.F., Fumarola, F., Ceci, M., Scarpino, A., Torelli, M., Malerba, D.: Automatic extraction of logical Web lists. In: Andreasen, T., Christiansen, H., Cubero, J.C., Raś, Z. (eds.) Foundations of Intelligent Systems, Lecture Notes in Computer Science, vol. 8502, pp. 365–374. Springer International Publishing (2014)
23. Lanotte, P.F., Fumarola, F., Malerba, D., Ceci, M.: Automatic generation of sitemaps based on navigation systems. In: Pardalos, P.M., Conca, P., Giuffrida, G., Nicosia, G. (eds.) Machine Learning, Optimization,

and Big Data - Second International Workshop, MOD 2016, Volterra, Italy, August 26-29, 2016, Revised Selected Papers, Lecture Notes in Computer Science, vol. 10122, pp. 216–223. Springer (2016)
24. Lee, U., Liu, Z., Cho, J.: Automatic identification of user goals in Web search. In: Proceedings of the 14th International Conference on World Wide Web, WWW '05, pp. 391–400. ACM, New York (2005)
25. Lie, H.W., Bos, B.: Cascading Style Sheets:Designing for the Web, 2nd Edition Addison-Wesley Professional (1999)
26. Lin, C.X., Yu, Y., Han, J., Liu, B.: 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, PAKDD'10, pp. 222–229. Springer, Berlin (2010)
27. Lin, S.H., Chu, K.P., Chiu, C.M.: Automatic sitemaps generation: Exploring website structures using block extraction and hyperlink analysis. Expert Syst. Appl. **38**(4), 3944–3958 (2011)
28. Liu, Z., Ng, W.K., Lim, E.P.: An automated algorithm for extracting Website skeleton. In: Database Systems for Advanced Applications, pp. 799–811. Springer (2004)
29. Luo, Y., She, G., Cheng, P., Xiong, Y.: Botgraph: Web bot detection based on sitemap. arXiv:1903.08074 (2019)
30. Mobasher, B.: Data mining for Web personalization. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web, Methods and Strategies of Web Personalization, Lecture Notes in Computer Science, vol. 4321, pp. 90–135. Springer (2007)
31. Mooney, C., Roddick, J.F.: Sequential pattern mining - approaches and algorithms. ACM Comput. Surv. **45**(2), 19:1–19:39 (2013)
32. Nakagawa, M., Mobasher, B.: A hybrid Web personalization model based on site connectivity. In: Proceedings of WebKDD, pp. 59–70 (2003)
33. Nemenyi, P.B.: Distribution-free Multiple Comparisons. Ph.D. Thesis, Princeton University, Princeton (1963)
34. Nielsen, J., Loranger, H.: Prioritizing Web usability. New riders publishing, thousand oaks, CA USA (2006)
35. Olston, C., Chi, E.H.: Scenttrails: Integrating browsing and searching on the web. ACM Trans. Comput.-hum Interact. **10**(3), 177–197 (2003)
36. Pons, P., Latapy, M.: Computing communities in large networks using random walks. Journal of Graph Algorithms and Applications **10**(2), 191–218 (2006)
37. Qi, X., Davison, B.D.: Web page classification: Features and algorithms. ACM Comput. Surv. **41**(2), 12:1–12:31 (2009)
38. Wang, J., Han, J., Li, C.: Frequent closed sequence mining without candidate maintenance. IEEE Trans. on Knowl. Data Eng. **19**, 1042–1056 (2007)
39. Wang, X., Ahuja, N., Llorens, N., Bansal, R., Dhar, S.: Toward an intelligent crawling scheduler for archiving news websites using reinforcement learning. Tech. rep., Virginia Tech. http://hdl.handle.net/10919/96482 (2019)
40. Weninger, T., Bisk, Y., Han, J.: Document-topic hierarchies from document graphs. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, CIKM '12, pp. 635–644. ACM, New York (2012)
41. Weninger, T., Han, J.: Exploring structure and content on the web: Extraction and integration of the semi-structured web. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining, WSDM '13, pp. 779–780. Association for Computing Machinery, New York (2013)
42. Weninger, T., Johnston, T.J., Han, J.: The parallel path framework for entity discovery on the web. TWEB **7**(3), 16:1–16:29 (2013)
43. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**, 80–83 (1945)
44. Yan, X., Han, J., Afshar, R.: Clospan: Mining Closed Sequential Patterns in Large Datasets. In: SDM, pp. 166–177 (2003)
45. Yang, C.C., Liu, N.: Web site topic-hierarchy generation based on link structure. J. Am. Soc. Inf. Sci. Technol. **60**(3), 495–508 (2009)
46. Yin, Z., Gupta, M., Weninger, T., Han, J.: A unified framework for link recommendation using random walks. In: 2010 International Conference on Advances in Social Networks Analysis and Mining, pp. 152–159 (2010)
47. Yujian, L., Bo, L.: A normalized levenshtein distance metric. IEEE Trans. Pattern Anal. Mach. Intell. **29**(6), 1091–1095 (2007)