Discovering structured information from websites

Pasqua Fabiana Lanotte
Dipartimento di Informatica
Dottorato in Informatica XXVIII ciclo
Università degli Studi di Bari "Aldo Moro"
Via E. Orabona, 4 - 70125 Bari, Italy
pasqua.lanotte@uniba.it

Supervisor: Prof. Michelangelo Ceci

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Computer Science

Bari, February 2017

Thesis Supervisor

Prof. Michelangelo Ceci

Chairperson of the Supervisory Committee

Member of the Supervisory Committee

Member of the Supervisory Committee

Submitted *February 2017*

Contents

List of Figures

List of Tables

Acknowledgments

There are many people I could thank, but time, space, and modesty compel me to stop here.

Abstract

The World Wide Web is the largest and most widely known repository of hypertext. Hypertextual documents (also called web documents or web pages) contain information about every topic and every real-world entity, authored and edited by millions of people and written in hundreds of languages. However, this repository can be considered as a modern legacy system because such stored data cannot be easily accessed and manipulated.

Web mining, that is the process of information discovery from sources across the World Wide Web, is one of the hottest topic into the Computer Science community. Although Web Mining borrows heavily from traditional fields such as Information Retrieval, Data Mining, Statistics, etc., the characteristics of the Web (e.g. heterogeneity, noise, dynamicity, dimension, etc.) make techniques and approaches used in these fields non directly applicable on data such as web pages or websites. Moreover, extracted information, in form of structured data, are very valuable both for improving the performances of existing applications (e.g. improving search engine results) and for generating new applications (e.g. automatic sitemaps generation, breadcrumb mining, etc.)

In my thesis, I investigate the principles and methodologies for extracting structured data from websites, merging structured data spanned on multiple web pages and organizing web pages based on structured data. This is achieved through the development of models and algorithms which exploit multiple web page representations (e.g. textual, visual, structural, etc.) and combine these information among them.

1 Introduction

1.1 Automated information gathering

The World Wide Web is the largest and most widely known repository of hypertext. Hypertextual documents (also called web documents or web pages) contain information about every topic and every real-world entity, authored and edited by millions of people and written in hundreds of languages. Moreover, hyperlinks creating billions of connections among web pages make the Web the largest and the most connected information network, where information in form of facts, ideas, and opinions are splitted and propagated among pages.

How to gather, extract and merge useful and meaningful information from the Web, however, becomes challenging to web users. In fact, access to web information is directly opposite to what we know from databases and libraries, where all data items or documents are well organized (in types, topics, area, years, etc.) and structured (e.g. tables).

Currently, there are two predominant and complementary paradigms for users to locate information: *i)* Keyword-based search and *ii)* Navigation paradigm [OC03, Fan07, Kal07, Lev10]. In the keyword-based search paradigm, the user types a search query (usually a list of key terms) into a search engine and obtains a ranked list of pages in descending order of relevance and accuracy respect to the input query. The main drawback of this paradigm is that users have to know what they are looking for. In fact, for expressing information needs to the search engine, users have to think of appropriate key terms. Therefore, this paradigm is useful when users are familiar with the search domain (e.g., when they know the best terms to express their information needs and discriminate as much as possible their search domain from other domains). In all the other cases, when for example users do not know what they are looking for un-

til the available options are presented or when their information needs cannot be formulated in keywords, this paradigm results unusable for the information gathering purpose [LLC05, OC03].

In the navigation paradigm, 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 (e.g. sitemaps, navbars, menus, etc.) provided by the website to find the desired information. The main drawbacks of this paradigm are the consuming time for finding specific information and the risk of get lost in the hyperspace. Getting lost comes from a feeling of frustration of not finding the information we are looking for and from a feeling of disorientation within the web site we are surfing through. To decrease this problem, several data mining tools are developed to increase the website usability and user interaction. Analogously to the diverse set of tools ranging from road signs, the compass and map, to global positioning systems (GPS), all of which we use to orient ourselves in physical spaces, these solutions allow web users to understand in the hyperspace where they are, where they can go and where they come from [NL06].

Gathering information from the Web is a challenging task also for machines. In fact, differently from web users, machines cannot easily understand the information codified in web pages. This is due by the fact that HTML code is used for data visualization rather than data representation. By this, we mean that humans can easily distinguish e.g. main and secondary menus, navbars, breadcumbs, product listing, advertisements, unstructured contents based on web page visual features and position. Machines in contrast cannot directly extract these semantics [KMH13]. To overcome this issue two main solutions are possible: *i)* using data markup to encode machine-readable knowledge (e.g. semantic formalization for structured data specified by Schema.org[1]); *ii)* novel data mining methods to extract structured information [LFC+14], to infer a structure or a schema for data codified in web pages [ZL05], and to facilitate data mash-up [DGH+14, BCMP13]. Then, information extracted by data mining tools can be used for helping web users in the information gathering process (e.g. clustering similar web pages and extracting the logical structure of websites such as sitemaps).

To summarize, the challenge of data mining tools for helping web users is to automate the information gathering process. For this scope, existing solutions

[1]Schema.org is an initiative of the major search engine operators for creating, maintaining, and promoting schemas for structured data on the Internet, on web pages, etc.. See `http://schema.org/`.

try to bring back the semantic and structure of the Web and its web pages in the way that users and machines can easily access and use the vast amount of information available. These solutions belong to several research fields, such as Web Mining [Che08](which analyze web pages' contents, hyperlinks and user logs), Web Information extraction [DMP12] (which extract information in form of structured data), Sequential pattern mining [MR13](for extracting recurrent patterns), Network Analysis [SH12] (which evaluates how information is propagated in the Web), etc. .

1.2 Mining the Web

The Web has many unique characteristics which make the discovering of novel and valuable information an interesting and challenging task [Liu06]. These characteristics can be summarized as follows:

- **Dimension**. The amount of data and information on the Web is huge and still growing. The Web is the first medium where the number of information producers, responsible for building without censure new facts, ideas, and opinions, is the same magnitude order of information consumers. Published datasets are so large and complex that traditional data processing applications are inadequate to deal with them. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualization, querying, updating and information privacy.

- **Dynamicity**. Each second thousands of web pages are created, destroyed and modified. This make the Web a dynamic information network, where the structure and the content change frequently. Keeping up with these changes and monitoring them are important issues for many applications.

- **Heterogeneity**. Web pages are heterogeneous in terms of formats and writing styles. In the first case, the heterogeneity is due by the fact that web pages do not respect a standard format. In fact, they can be classified in three categories [CKGS06]: i) unstructured pages; ii) structured pages; iii) semi-structured pages.
  Unstructured pages, also called free-text documents, are written in natural language. No structure can be found, and only information extraction (IE) techniques can be applied with a certain degree of confidence [Sar08].
  Structured pages are normally obtained from structured data sources (e.g., databases) and data are published together with their schema. In general,

information extraction on structured pages is accomplished using techniques based on syntactic matching.

Semi-structured pages are in an intermediate position between unstructured and structured pages. These documents possess anyway a kind of structure which is enclose in free-text. Extraction techniques are often based on the presence of regular patterns as HTML tags, CSS code, etc. [FMFB12].

Heterogeneity of web pages is also due by the presence of different and informal writing styles [Liu06, KC12]. In fact, differently from a traditional textual collection, web pages are created by millions of people having different cultures, skills, languages, etc.. This means that web pages may present the same or similar information using completely different words and/or formats. This makes extraction and integration of information from multiple pages a challenging problem.

- **Connection**. The Web is generally represented as an information network, that is a graph, where nodes are web pages and edges are hyperlinks. Hyperlinks in the Web have several roles and functionality. In particular, hyperlinks between pages belonging to the same website (e.g., domain) codify the navigation systems of the website (e.g., menus, navbars, sitemaps, etc.). In other words, they serve as mechanisms to coherently organize the information within the website and assist users during the website navigation. Differently from hyperlinks belonging to a single website, hyperlinks across different websites encode latent human judgments of authority to the target pages [Lev10]. In this case, a link represents a concrete indication of the following type of judgment: the creator of page p, by including a link to page q, has in some measure conferred authority (i.e., trust) on q. According to this idea, it is possible identify in the information network *authoritative pages*, which provide good information, and *hub pages*, which provide links to good authorities pages.

- **Noise**. The richness of information has also made the Web progressively more difficult to leverage the value of information. Differently from other media, information publication in the Web does not require editorship and approval from some authority. This unregulated atmosphere contributes not only to the big volume and wide diversity of information, yet also to the presence of low quality, misleading, erroneous, and redundant data. Moreover, noise comes from another source. A typical Web page, due its rich semantic, contains multiple pieces of information organized and visu-

alized in the way that users can easily recognize each of those, e.g. the main content, navigation links, advertisements, copyright notices, privacy policies etc. [KN12]. For particular applications, only a part of information is useful (e.g. main content or navigational links) while the rest is considered noise [YLL03, Liu06, KN12]. Therefore, applications which focus on subsets of information stored in web pages, should do not consider a web page as atomic node in the web graph, but they should identify web page's information blocks as atomic units. In this context, an information block is a subset of a web page where web elements in the block have similar functionality and similar visual and structural properties [LCC11].

- **Virtual society**. The Web may be considered a large social network where people can communicate and influence other people. In fact, the Web is not only about data, information and services, but also about interaction among people, organization and automated systems [GHFZ13].

The previous features of the Web present both challenges and opportunities for extraction and mining of information and knowledge from the Web. In this context, the aim of the Web Mining is to discover useful information or knowledge from the *web hyperlink structure*, *page content*, and *usage data* (e.g., web server access logs, user profiles, user queries and click-stream). Although it borrows heavily from traditional fields such as Information Retrieval, Data Mining, Statistics, etc., the characteristics of the Web make techniques and approaches used these fields non directly applicable on data such as web pages or websites.

Web Mining algorithms can be classified in three main categories based on the type of data used for the mining process:

- **Web Structure Mining**. It extracts previously unknown relationships among web pages (ranging from a single website to the web as a whole) analyzing the hyperlinks structure of the Web (also called web graph). The analysis of hyperlinks allows us to understand the overall websites structure and discover the information flow (e.g., where information is concentrated or missing, and ho the information is propagated). Examples of web structure mining algorithms are clustering of connected web pages having a similar template [Got08, BDM11], discovering of authoritative web pages [Kle99, BP12], extraction websites hierarchies [LCC11, WBH12], etc.. Traditional Data Mining does not perform such tasks because there is usually no link structure in a relational table.

- **Web content mining**. It extracts or mines useful information or knowl-

edge from web page contents. Examples of Web content mining algorithms are automatic classification or clustering of web pages according to their topics, automatic extraction of recurrent patterns in web pages such as descriptions of products, postings of forums, product listing, etc.. Although these algorithms can appear similar to traditional Data Mining or Text Mining algorithms, characteristics of web pages (e.g., presence of HTML tags, CSS code, etc.) make algorithms belonging to these fields non directly applicable on web pages. An important sub-field of the Web Content Mining is the Web Information Extraction which goal is to extract structured data from web pages and map these data in relational tables (see Sec. 1.4).

- **Web usage mining**. It refers to the automatic discovery and analysis of patterns in click-stream and associated data collected or generated as a result of user interactions with web resources on one or more websites. The discovered patterns are usually represented as collections of pages, objects, or resources that are frequently accessed by groups of users with common needs or interests. Web usage mining applies many data mining algorithms on web logs data properly collected and pre-processed. The major application areas for Web usage mining are personalization, system improvement, site modification, business intelligence, and usage characterization [SCDT00].

1.3 The role of structured data in the Web

A large amount of information from the Web is represent in form of semi-structured data, that is a combination of unstructured text with data having a structure or a schema [AGM02]. Structured data contained in web pages are typically **data records** generated dynamically from an underlying structured source like a relational database (e.g., product listing of an Amazon web page) or from a static template (e.g., menus, navbars, etc.).

Figure 1.1 shows an Amazon web page returned for the query *Computer*. In such page it is possible identify several groups of structural data. In particular, links in structured data that describe the product listing (i.e., box A) allow us to obtain key information related to real-world entities (e.g., computers) while links contained in the other structured data (boxes B, C, D, E, F) allow us to navigate the organization of the website and identify meaningful connections among objects (e.g., identify computers belongs to the same brand).

Extracting such data records is useful in several application domains because

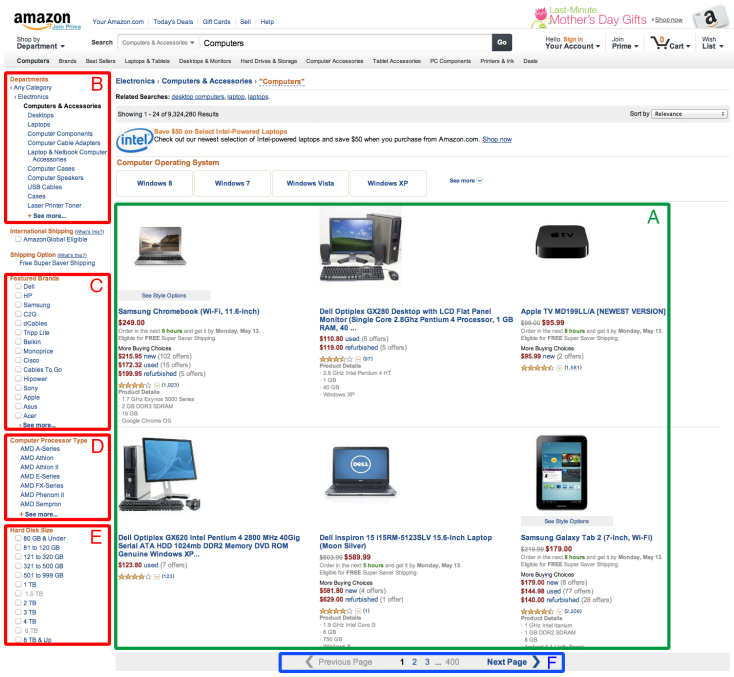


FIGURE 1.1: An example of Amazon Web page

it enables us to obtain and integrate data from multiple sources (websites and web pages) to provide value-added services, e.g., customizable web information gathering, comparative shopping, meta-search, etc. With more and more companies and organizations disseminating information on the Web, the ability to extract such data from web pages is becoming increasingly important.

Although these data are easily identified by humans, machines are not able to directly access to structured data encoded in web pages. Moreover, since web documents are neither well structured such as database nor completely unstructured such as pure textual documents, traditional Data Mining or Text Mining techniques can not be directly applied on structured data encoded in web pages.

In the first case, Data Mining techniques are based on the assumption that data used for learning models share a common schema having well defined tables, attributes (columns), tuples (rows), keys, and constraints. Moreover, such data should be independent among them. Web pages break this assumption because structured data do not share a common schema and their hyperlinks define interdependence relationships among web records and web pages. Moreover, web pages are codified in HTML a markup language that, differently from other language markups used to store data such as XML, was projected just for data

rendering. Consequently, the Web can be considered a modern legacy system because such a large body of data cannot be easily accessed and manipulated.

In the second case, Text Mining techniques extract structured data in web pages analyzing recurrent patterns and named entities in word sequences [Sar08]. Existing solutions for plain documents fail to learn accurate models because they are based on assumption that the document collection is written with a consistent writing style (e.g., news articles). Moreover, text mining approaches, considering documents as sequences of words, are not able to extract valuable information from different web page's representations. In fact, differently from textual documents, web pages have multiple representations which provide different information; one is the text representation written in HTML; the other is the visual representation rendered by a web browser. These sources of information are completely ignored by Text Mining approaches. As consequence, they are not able to handle complex information within web elements possessing various semantic roles (e.g., a navigation menu, main content, calendar, table, and logotype) and providing different functionalities (e.g., a link, button, and element with drag-and-drop function) [QD09]. Finally differently from textual documents, web pages are enriched by hyperlinks that enable information to be splitted in multiple and interdependent web pages. These hyperlinks can be used to identify collections real-world entities (e.g., web pages of courses, professors, products, news, etc) and relationships among entities. Also this information is ignored by Text Mining.

Consequently, there is a strong need in the computer science field of creating techniques and approaches that, using textual, structural, and visual information of web pages, are able to extract schema from structured data and align the data using that schema. Goal of Web Information Extraction is that to transform the Web from a legacy system to the biggest, structured, and easily accessible database.

1.4 The role of Information Extraction

Information Extraction born originally as a natural language processing task used to extract relevant information both from structured text with tabular information and from free text such as news articles [Eik99]. Goal of Information Extraction is to identify patterns involving syntactic relationships between words or semantic classes of words. Extracted patterns can be used generate a relation of k-tuple (where k is the number of attributes in a record) or a complex object with hierarchically organized data [CKGS06]. An application example

is to extract from a terrorist attack article key information about perpetrators, their affiliation, victims, location, etc [PLCM14].

With the growth of the Web researches tried to apply information extraction techniques on web pages. As said in the previous section, web pages differ from textual documents for several aspects, such as absence grammatical structures, presence of hyperlinks, etc. For this reason, traditional information extraction tools which use linguistic knowledge to extract key information are not suitable for web pages. For example, most of structured data in web pages are rendered using regular HTML tags patterns and information are splitted in interconnected web pages. For these web pages the analysis of regularity on their visual and structural representation and the analysis of the graph behind the website can be used for extracting structured data more accurately and efficiently than natural language processing methods.

Information extraction systems on the Web require to solve five distinct problems:

- **Navigation problem**: finding target web pages, i.e., pages containing data to extract in a website following hyperlinks. Websites, especially data intensive websites, contain both target pages and navigational pages (i.e., web pages contain hyperlinks to target pages or to other navigational pages). A system for extracting structured data should explore web pages in a way to minimize the search space to target pages and as less navigation pages as possible.

- **Data extraction problem**: extracting relevant data records from web pages. Solutions to extract data records should analyze structural and visual properties of web elements to find recurrent patterns which describe data records.

- **Schema synthesis problem**: generating schema behind extracted data-records. In this case solutions are needed to extract and align attributes from data records. These solutions should be able to handle missing values, values embedded in plain text paragraphs, values hierarchically organized, etc.

- **Data mapping**: aggregating data from several websites. This requires data be homogenized since different websites can follow different conventions for naming things, for expressing measured units (e.g. currency, weight of a product, etc.), etc.. Mapping discrete values (e.g. company names) into a standard format and transforms measured values in a common unit improves the quality of the extracted data.

- **Data integration**: merging data from separate web pages. Some websites, especially websites with a large amount of data (e.g. Ebay, Amazon, etc.), split structured data on multiple web pages for avoiding to overload a single page with too much information (e.g., splitting the products listing using the pagination list). Other times information about a single data record are splitted on multiple web pages (e.g., reviews of a product and main information about the product self are stored on two different web pages). Large scale programs able to fetch tens of thousands of web pages per second are called *crawler, spiders, web robots, or bots.*

To extract structured data from the Web, several types of wrappers have been created. A wrapper can be defined as a program that extracts structured data of a particular information source and translates them into a relational form [Kus97]. Existing wrappers can be categorized in three main groups:

- **Manual wrapper**: it involves the writing of ad hoc code. Human programmers identify syntactic patterns to extract structured data analyzing source codes (i.e. HTML code) of web pages and understanding their structure. Although this approach is simple, it suffers several disadvantages due Web features. The first drawback is due by heterogeneity and dimension of the Web; manual wrappers are not scalable because websites and web pages with different structures require different wrappers. Moreover, the Web is a dynamic environment where web pages are created, destroyed and modified; wrappers can not automatically adapt to these changes since they are based on a specified grammar and manual maintenance has high cost.

- **wrapper induction**: it consists in the automatic wrappers construction based on inductive learning methods. The first wrappers were created around 1995-1996. In this case, supervised learning approaches are used to automatically generate rules which are then used to extract structured data from unseen web pages. Wrappers based on induction learning can be classified as *zero-order* or *first-order* depending on the inductive learning method used [Eik99]. Wrappers constructed using zero-order learning methods learn models in form of couples *attribute-values*, that is, from database point of view as disconnected relations. Differently from zero-order wrappers, first-order wrappers are able to learn models in form of first-order predicates which describe relations and associations between these relations.

  The accuracy of learned rules, and consequently the accuracy of this type

of wrappers, strongly depends both the number and the quality of examples (i.e. manually labeled pages and data records). Wrappers generated using supervised learning require a high cost for manual labeling of examples, for generating wrappers from websites with different structure, and for the maintenance in case changes in the websites structure.

- **automatic extraction**: it consists in the automatic wrappers construction based on unsupervised learning. Differently from the previous approaches it is not required the manual labeling effort. This makes wrappers scalable to a large amount of websites and easily maintainable. Modern wrappers are based on this approach.

1.5 The role of information network analysis

Nowadays, most real phenomena can be represented as information networks, that is graphs where the nodes are real-world entities interconnected and interacting among them. Formally an information network can be defined as follow [SH12]:

Definition 1 ***Information Network:*** *is defined as a directed graph* $G = (V, E)$ *with an object type mapping function* $\tau : V \rightarrow A$ *and a link type mapping function* $\phi : E \rightarrow R$*, where each object* $v \in V$ *belongs to one particular object type* $\tau(v) \in A$ *, each link* $e \in E$ *belongs to a particular relation* $\phi(e) \in R$ *, and if two links belong to the same relation type, the two links share the same starting object type as well as the ending object type.*

FIGURE 1.2: An example of an information network of a website where nodes are web pages related to different entity types (e.g. people, news, organizations) and edges are hyperlinks

The analysis of information networks has gained extremely wide attentions nowadays from researchers in several fields such as computer science, social

science, physics, economics, bioinformatics, and so on, with exciting discoveries and successful applications across all the disciplines. Goal of information network analysis is to extract patterns or regularities in relationships among interacting units which represent the structure of the network.

Figure 1.3: An example of a topological structure hidden into the information network of a website

For example, the application of data mining or graph mining techniques on information networks allows researches to discover latent structures among nodes, such as the topics and communities they are involved in, the roles that different types of nodes play in these topics and communities, and the relations they potentially have with each other. These approaches can be used in the context of the Web for combining intra-page information (e.g. textual content, HTML structure) with inter-page information which describe the topological structure of websites.

Although several methods and approaches exist for extracting knowledge from the information networks, most of the existing studies are based on the assumption that networks are homogeneous. However, real information networks contain heterogeneous and structured data (e.g., relational database data) [SH12]. Although a single link in the network could be noisy, unreliable and sometimes misleading, valuable knowledge about the structure of networks can be discovered reliably analyzing massive connections between objects [KHY+08].

1.6 Outline of the thesis

In this thesis I investigate the principles and methodologies for discovering novel and valuable knowledge from the Web. I propose models and algorithms which exploit the main features of web pages and websites for extract structured in-

formation encoded in one or more web pages or hidden in the link structure of websites. Proposed methods involve several fields, ranging from Web Information Extraction to Information Network Analysis, Sequential Pattern Mining, Web Structure and Content Mining. Moreover, the methods and approaches analyzed and implemented in this thesis contribute to give a structure to the Web exploiting and discovering properties and interactions among web pages that were previously unknown. Therefore, extracted information can improve the information gathering process both for humans and for machines. This is realized analyzing and combining different web page representations (e.g. textual representation, visual representation, structural representation, etc.).

The major contributions are threefold. First, I propose a novel solution to extract structured data in form of *web lists* splitted on multiple web pages. Although this task has been studied extensively, existing approaches are based on the assumption that lists are wholly contained in a web page. They do not consider that many websites span structured data of same semantic class (e.g. books, professors, product listing) on several web pages and show for each of these only a partial view. Proposed method combine information intra-page (i.e web lists) and extra-page information (website structure) to extract a complete collection of structured data belonging to the same semantic class.

Second, a method to automatically discover sitemaps is realized. Actually sitemaps are manually generated by web designer or automatically extracted by information network analysis tools. In the former, manual approaches extract static and deeper (with respect to automatic methods) hierarchies which are not able to capture evolutions in the website (e.g. generation of new sections, deletion of existing web pages, etc.). This makes sitemaps helpless and confusing for users after few time. In the latter, existing solution extract only a flat list of website's urls that do not show the hierarchical structure of a website or use only web pages' content ignoring the website's link structure. Differently from existing automatic solutions, the proposed approach is both automatic and effective. It explores navigation systems (e.g. menu, navbar, content list, etc.) contained in a website and exploits recurrent patterns of navigation systems to discover rich hierarchies that unveil relationships among web pages (e.g. relationships of super/sub category). For this scope, a novel sequential *closed* pattern mining algorithm, called CloFAST, is implemented. CloFAST combines a new data representation of a sequence dataset with a novel one-step technique to both generate sequences and to prune the search space. These features allows CloFAST to drastically reduce the number of intermediate sub-sequences and then reduce the computational complexity of the sequence generation process

while preserving the same expressive power of patterns extracted by means of existing algorithms. Reducing the computational complexity of algorithms used the Web context is fundamental task.

Third, I focus on another challenging task in Web Mining: clustering of web pages. Web pages are characterized by different roles and several representations, based on textual, hyperlink and HTML formatting (i.e. HTML tags and visual) properties. Existing clustering algorithms use these information almost independently, mainly because it is difficult to combine them. The proposed solution is intended to be a contribution on clustering of web pages in a website by combining all this features into a single vector space representation.

2

Automatic Extraction of Logical Web Lists

2.1 Introduction

The Web contains a large amount of structured data, most of which represent the main content of web pages. For example, given a Amazon web page that describes a DVD product, provided information about the item, such as prize, title, reviewers, ads, etc. are organized in a structured way.

Although humans can easily understand how such information are structured (e.g., distinguish between the structured data and the rest of the web content), machines cannot extract this semantic from the HTML code. Nowadays, to solve this issue two main approaches are applied:

- Using data markups to encode machine-readable knowledge. These markups are not related to the formatting of data but they just make the metadata and text enclosed within the XHTML tags more meaningful to computers.
- Data Mining methods automatically extract structural data based on structural features, visual features or both.

Although using data markups makes web pages machine-readable, there are several reasons why the first solution can not be applied on all websites. First, data intensive websites such as Deep Web Databases (e.g. Amazon.com, Trulia.com) are not favorable to make accessible all their information assets. Another important reason is that enriching web pages with XHTML tags requires human efforts which do not involve direct earnings for organizations' websites.
Several methods to extract structured data have been presented in the literature. The first approaches were hand-crafted web scrapers based on regular

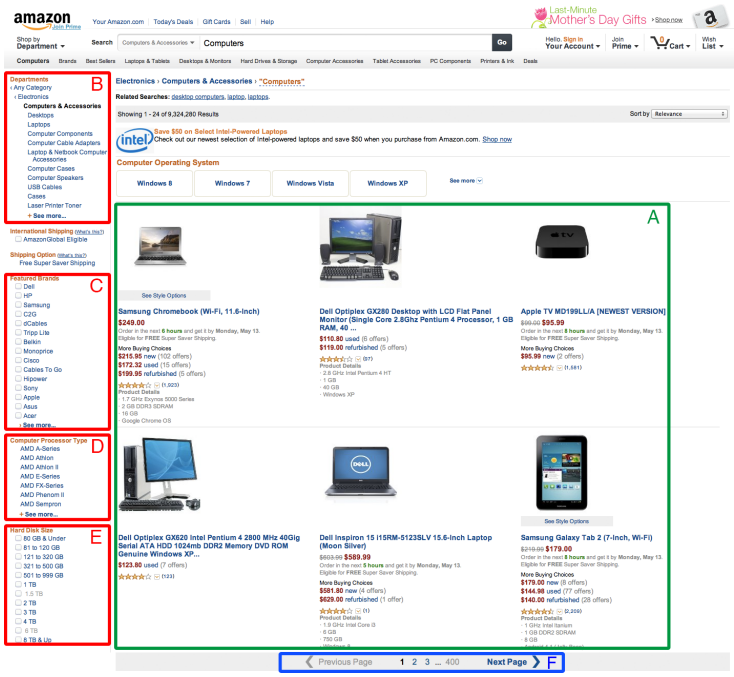


Figure 2.1: An example of Amazon web page

expressions. An obvious disadvantage of these approaches is that different rule expressions need to be manually created for each website. Furthermore, an individual website may also change its structure or layout over time making this approach not scalable and in need of continuous maintenance.

Several supervised and unsupervised Data Mining methods were implemented later [LGZ04, MTH09, LGMK04, GBH+07, LMM10]. Although these methods close the challenge of automatic extraction of structured data from a web page, they fail to detect data record which span multiple web pages. This is an open issue, because many websites, especially data-intensive (e.g. Amazon, Trulia, AbeBooks,...), present their listings as *logical list*, that is, a list spanning multiple pages (*e.g.* computers, books, home listings)[1]. It is as if, each list represents a *view* of the same *logical list*. Similar to databases, where a *view* can represent a subset of the data contained in a table partitioned over a set of attributes, a *logical list* is split in multiple *views* (web pages) in order to avoid information overload and to facilitate users' navigation.

For example, Fig. 2.1 shows a web page from Amazon.com that contains the results for the query "Computer". On this page, the boxes A, B, C, D, E, F

[1]The motivations behind this approach are as well technical (reducing bandwidth and latency), and non technical as avoiding information overload or maximizing page views.

are web lists. The list in the box A shows a *view* of the "Computers" products, that is the top six sorted by relevance, and F allows us to navigate to the other views of the products ordered by relevance. Thus navigating the links in F we can generate the *logical list* of the products for the query "Computer". Boxes B, C, D and E contain respectively the lists representing filters for "Department", "Featured Brands", "Computer Processor Type", and "Hard Disk Size", which are attributes of the *virtual* table "Computer". Moreover, the anchor-text links in boxes B, C, D and E stores valuable information which can be used to annotate data records, and thus to individuate new attributes. For example, the anchor-text links of web list C can be used to index data records based on "Computer brands". Traditionally, search engines use the proximity of terms on a page as a signal of relatedness; in this case the computer brand terms are highly related to some data records, even though they are distant.

Providing automated techniques for *logical list* extraction would be a significant advantage for data extraction and indexing services. Existing data record extraction methods [LGZ04, MTH09, GBH+07, LMM10] focus only in extracting *view* lists, while several commercial solutions[2] provide hand-coded rules to extract *logical lists*.

In this chapter, I face this issue by proposing a novel unsupervised algorithm for automatic discovery and extraction of *logical lists* from the Web. This method requires only one page containing a *view list*, and it is able to automatically extract the *logical list* containing the example *view list*. Moreover, during the process, it enriches the list's elements with the pair *<url, anchor-text>* used for the extraction task. We have validated our method on a several real websites, obtaining high effectiveness.

2.2 Definitions and Problem Formulation

In this section, I introduce a set of definitions that will be used through the thesis.

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). For extracting logical lists both the visual and the structural representations are exploited.

Definition 2 *A web page is characterized by a* ***Structural Representation*** *composed by web elements inscribed in HTML tags and organized in a tree-based*

[2] Lixto, Screen Scraper Studio, Mozenda Screen Scaper

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 3 *The* ***Web Page Rendering*** *is the process of laying out a spatial position of all the text/images and other web elements in a web page to be rendered.*

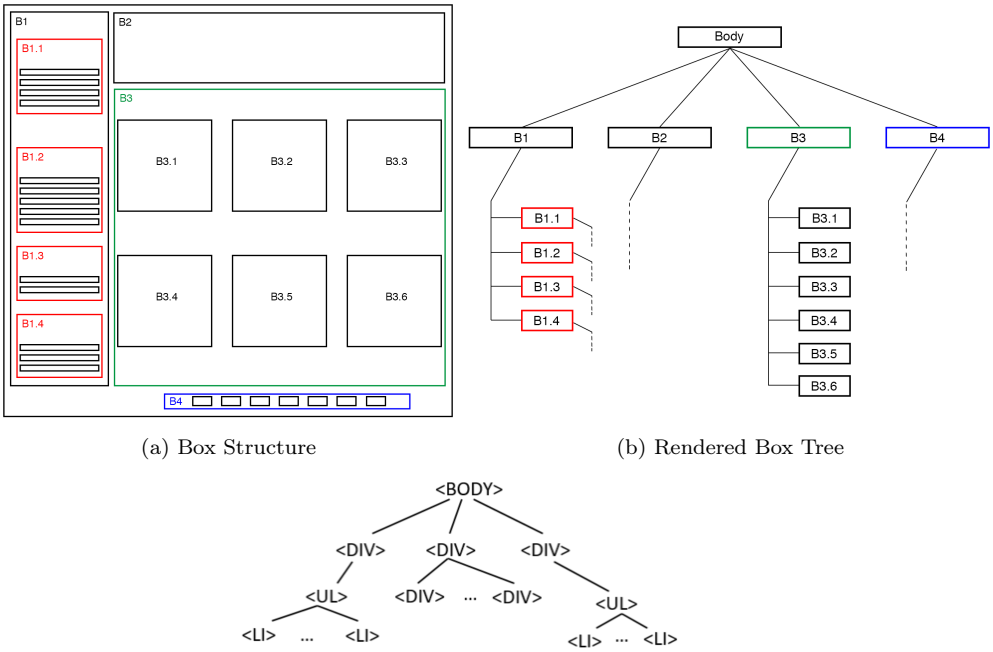
Definition 4 ***Web Page Visual representation.*** *When a web page is rendered in a web browser (see Def. 3), the CSS2 visual formatting model [LB99] represents the web page's elements by rectangular boxes that are laid out one after the other or nested inside each other by forming a tree, called* ***Rendered 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 [LB99]. 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 2.2 shows an example of the structural and visual representations for the Amazon web page in Fig. 2.1. 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 2.2(b) shows (there are often hundreds or even thousands of blocks in a Rendered Block Tree).

Both Definition 2 and Definition 4, which are used to exploit the web page stucture, are used in the definition of web lists, which is crucial for the task of logical list extraction:

Definition 5 *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) [LFC+14].*



(a) Box Structure

(b) Rendered Box Tree

(c) Structural Representation

Figure 2.2

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 2.3.

From the Amazon web page, showed in Fig. 2.1 it is possible to extract one tiled list (i.e., box A), one horizontal list (i.e., box F) and four vertical lists (i.e., boxed B, C, D, E).

Definition 6 *A **Data Record** identifies an element of a web list. Similar to the concept of data records into database, data records into a web page are a set of similar and structured objects containing information related to a real-world entity.*

Definition 7 ***Logical List**: It is a list whose Data Records are distributed on more then one web pages.*

An example is shown in Fig. 2.3, where the boxes A1 and A2 represent a part of a *logical list*.

Definition 8 ***View List**: It is a view of a logical list, whose Data Records are all contained in same web page.*

List F in Fig. 2.1 is an example of a view list. In fact, it contains only some of data records belonging to its logical list (that is the *pagination list*).

Definition 9 ***Dominant List:** It is the view list of interest, containing data records from the logical list that we want to extract.*

The choose of the dominant list strictly depends on the application goals. For example, if we want extract the whole product listing in an e-commerce website returned by a query, the dominant list is the web list containing products self (i.e. box A in Fig. 2.1). However, we could be interested to the navigation systems of a web page. In that case our dominant list could be for example its pagination list (i.e., box F in Fig. 2.1).

2.3 Methodology

In this section, I describe the methodology used for *logical list* extraction process. The algorithm employs a three-step strategy. Let P a web page, it first extracts the set L^P of the lists contained in P; in the second step, it identifies the *dominant list* $l^P_{dom} \in L$; finally, it uses l^P_{dom} to discover the *logical list* LL which includes l^P_{dom} as sub-list. These steps are detailed in the following sub-sections.

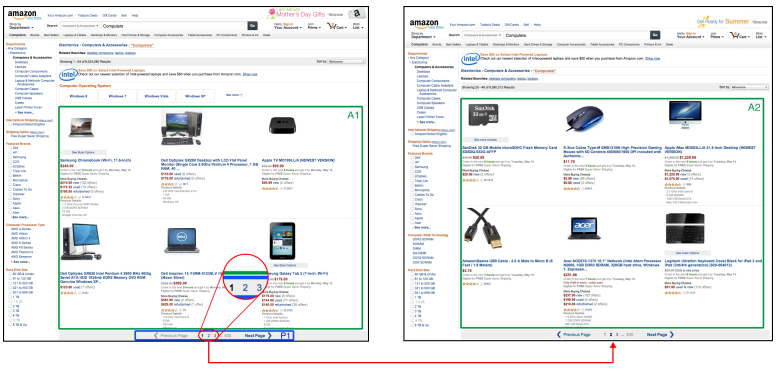


FIGURE 2.3: An example of *logical list* for Amazon's products.

2.3.1 List Extraction

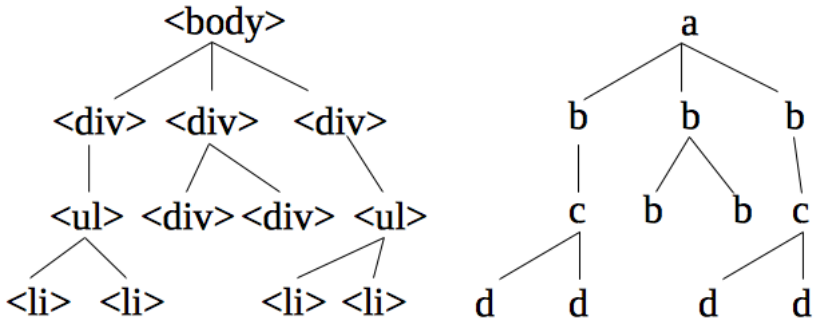
Given a web page P as input, its web lists $L = \{l_1, l_2, \ldots, l_n\}$ are extracted. According to Definition 5, Algorithm 1 describes in detail the process of web list extraction. Before to analyze the implementative aspects of this algorithm we need to explore the concepts of *structural similarity*, *adjacency* and *alignment*, between web elements, mentioned in Def. 5.

To check whether two web elements e_i, e_j have similar HTML structure several distance measures can be applied. In the following we describe the measures used, throw this thesis, for computing the structural similarity among web elements:

- *Normalized Edit Distance.* It measures the difference between two strings (or sequences) in terms of the minimum cost needed to transform a string A into the string B (through inserting, deleting or updating operations). This measure is then normalized for the maximum strings' length.

$$NormalizedEditDistance(A, B) = \frac{EditDistance(A, B)}{max(|A|, |B|)} \tag{2.1}$$

  For a survey on the Normalized Edit Distance and its variances see [YB07]. For applying the normalized edit distance to two HTML tag trees (web elements) e_i, e_j, a conversion step to translate each HTML tag tree into a string format is required. For this scope, each HTML tag in the tree rooted at $e_i(e_j)$ is first codified into a unique character. Then, a string is extracted from the converted tree by applying a breadth search on its nodes. Figure 2.4 shows an example of an HTML tag tree rooted at tag $< body >$ and its converted tree. Applying a breadth search to the converted tree, we obtain the string *abbbcbbcdddd*. Normalized edit distance



(a) Example of an HTML tag tree

(b) Example of a converted HTML tag tree

Figure 2.4

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 structural representation, the normalized edit distance can be replaced by the *Normalized Tree Edit Distance*

- *Normalized Tree Edit Distance.* Similar to the 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. Although tree edit distance algorithms are in general computationally more expensive than string edit distance algorithms, they are more accurate to extract web lists whose data records have deep and complex structural representations. For a survey on the Normalized Tree Edit Distance and its variances see [DMRW09]. For the scope of this thesis I have implemented the Tree Edit Distance proposed in [ZL05] and I have normalized it with the maximum number of nodes contained in A or B.

$$NormalizedTreeEditDistance(A,B) = \frac{TreeEditDistance(A,B)}{max(|A|,|B|)} \quad (2.2)$$

In addition to the structural similarity, other concepts required for the task of web list extraction are those of adjacency and alignment. In particular, 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, 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 $e_i.y = e_j.y$.

After having defined the concepts of structural similarity, adjacency and alignment, we can describe the method *extractWebLists()* (Algorithm **??**). In particular, the algorithm starts analyzing all the nodes which, in the rendered box tree, are children of the rectangular box representing the *body tag* element (line 2). For each node to analyze (line 4), only if the number of nodes in the subtree rooted in that node is relatively small, the algorithm tries to align its children (lines 9-30). 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. For the experiments, the threshold value we use for the size of the tree (*maxSize*) is 60. Only the nodes which are included neither in *verticalLists* (which contains vertically aligned web elements) nor in *horizontalLists* (which contains horizontally aligned web elements) are further explored by the algorithm (lines 27-28). The method *getStructurallySimilar()* is the method that checks whether the elements in a list have similar HTML structure. In this chapter the normalized tree edit distance is applied.

Algorithm 1 extractWebLists(P)

Input: web page P;

Output: List<List<Web elements>> L; //list of weblists.

```
1: body = webpage. findElementByTagName("body");
2: notAligned = body. getChildren();
3: repeat
4:   node = notAligned. dequeue();
5:   children = node. getChildren();
6:   if node.getSize() >= maxSize then
7:     notAligned. enqueue(children);
8:   else
9:     verticalAligned= children.groupByX();
10:    horizontalAligned = children.groupByY();
11:    for each list ∈ verticalAligned do
12:      structurallySimilar = getStructurallySimilar(list);
13:      if structurallySimilar.getSize() ≥ 2 then
14:        L.add(structurallySimilar);
15:      end if
16:    end for
17:    for each list ∈ horizontalAligned do
18:      structurallySimilar = getStructurallySimilar(list);
19:      if structurallySimilar.getSize() ≥ 2 then
20:        L.add(structurallySimilar);
21:      end if
22:    end for
23:    notIncluded = children - lists.getWebElements();
24:    notAligned. enqueue(notIncluded);
25:  end if
26: until !notAligned.empty() return L
```

2.3.2 Dominant List Identification

Given a web page P and the set of list $L = \{l_1, l_2, \ldots, l_n\}$ extracted in the first step, three measures are used to identify the *dominant list* of P:

- **Centrality.** Given a list $l_i \in L$, the *centrality* of l_i w.r.t P is obtained by computing the Euclidean distance between the center of the parent-box of l_i and the center of root-box of P.
- **Area Ratio.** Given a list $l_i \in L$, the *area ratio* of l_i w.r.t P is the size of

the box containing l_i divided the size of root-box of P.

- **Text-Tag Ratio.** Given a list $l_i \in L$, and let m the length of l_i, the *text-tag ratio* of l_i is computed as:

$$\frac{1}{m}\sum_{j=0}^{m}\frac{chars(l_i[j])}{tag(l_i[j])} \tag{2.3}$$

where $tag(l_i[j])$ is the number of HTML tags contained in the j-th data record of l_i and $chars(l_i[j])$ is the total number of characters contained in $l_i[j]$. Before that the text-tag ratio is computed, *script* and *remark* tags are removed because this information should be not considered in the count of non-tag text.

In particular the *Dominant list* of P is the list with the highest sum of contributions:

$$\arg\max_{l_i \in L} \frac{\alpha_1}{centrality(l_i)} + \alpha_2 areaRatio(l_i) + \alpha_3 textTagRatio(l_i) \tag{2.4}$$

where $centrality(l_i), areaRatio(l_i)$ and $textTagRatio(l_i)$ are respectively the centrality measure, area ratio and text-tag ratio of a list l_i contained in L. $\alpha_1 = \alpha_2 = \alpha_3$ are set to 0.3 to give the same weight to each measure.

Algorithm 2 `LogicalListDiscovery`

Input: dominant list l^P_{dom}, set $L_- = \{L \setminus l^P_{dom}\}$
Output: logical list LL

1: $LL = \{l^P_{dom}\}$
2: **for all** $l \in L_-$ **do**
3: **for all** $u \in l$ **do**
4: $L_u \leftarrow$ HyLiEn(u)
5: L_u.filterSimilarity(l^P_{dom}, α)
6: LL.add(L_u)
7: **end for**
8: **end for**
9: $LL \leftarrow LL$.flatMap()
10: $LL \leftarrow LL$.removeDuplicates()
11: **return** LL

2.3.3 Logical List Discovery

Identified the dominant list l^P_{dom} of the web page P, the last step of the algorithm is to discover the logical list LL containing l^P_{dom}. This is done by taking advantage of the regularities of wesites. As described by Crescenzi et al. [CMM05],

web page links reflect the regularity of the web page structure. In other words, links that are grouped in collections with a uniform layout and presentation usually lead to similar pages. Link-based approaches are used in the literature for tasks strictly related to the one solved by our method. For instance, Lin et al. [LZW+10] used web links to discover new attributes for web tables by exploring hyperlinks inside web tables. Lerman et al. [LGMK04] uses out-links to "detail web pages" in order to segment web tables. In this champter, I successfully use links grouped as lists to navigate web pages and to discover *logical lists*.

Algorithm 2 describes the approach used. It takes as input the dominant list l^P_{dom}, the minimum similarity threshold α, and the set of the lists L extracted from P. It iterates over all the lists in the set $L_- = \{L \setminus l^P_{dom}\}$ (line 1), and, for each url u in l_i it alternates, (i) the extraction of the set list L_u contained in the web page U having u as url (line 4) to, (ii) the filtering of all the lists in L_u which have a similarity with l^P_{dom} lower than α (line 6). At each iteration, all the lists resulting from step (ii) are added to LL (line 7). Finally, LL is flattened and all the duplicate elements are merged (lines 8-9). Moreover, during the process all the anchor text of url u are used as attributes to annotate the discovered *view* lists and are reported in the final *logical list* LL.

2.4 Experiments

In this section I present the empirical evaluation of the proposed algorithm. For this scope, a test dataset is manually generated and verified. In particular, for the experiment, I select 40 websites in different application domains (music shops, web journals, movies information, home listings, computer accessories, etc.) with list elements presented in different ways. For the deep-web databases, I performed a query for each of them and collected the first page of the results list; for others website I manually selected a random web page. Table 1 shows in the first column the ground truth, that is, the number of data records which belong to the *logical list* to be extracted. The dataset is composed of 66.061 list elements extracted from 4405 web pages. In particular, the following steps are carried out to extract from an input page its logical list: (i) the web page is rendered in a web browser, (i) the *dominant list* is manually identified and its data records are included in the logical list, and (ii) the *logical list* is enriched following the out-links of the other lists in the page. This task required around 7 days of 4 people.

To the best of my knowledge the task of *Logical List Discovery* is novel,

and there are not any other methods to compare with. So, I evaluated the effectiveness of our algorithm by using *precision*, *recall* and *f-measure* metrics, computed over the number of *logical list* elements to be extracted (manually verified) w.r.t to the algorithm results. In particular, the precision is the measure of, how many of the extracted *view* lists belong to a *logical list*. The recall allows us to measure how many of the discovered *view* lists are true positive element of a *logical list*. i also included the *F-Measure* which is the weighted harmonic means of *precision* and *recall*. These metrics are evaluated counting how many data records of the *logical list* are found in the *view* lists.

$$precision = \frac{TP}{TP + FP},\ recall = \frac{TP}{TP + FN}, F{-}measure = \frac{2(precision \times recall)}{precision + recall} \quad (2.5)$$

2.4.1 Results

The execution of the algorithm requires two parameters which are empirically set to $\alpha = 0.6$ and $\beta = 50$. These parameters are need by the HyLiEn Algorithm. Our methods uses α during the *Logical List Discovery* step.

Table 1 presents the main results. The first column holds for each logical list the number of data records to extract. The second and the third columns contain the number of true positive and the number of false negatives data records. I do not plot the number of false positives, because our algorithm outputted always 0 false positives during the experiment evaluation. Finally, the fourth, fifth and sixth columns show the values for precision, recall and f-measure.

In general, the experimental results show that our algorithm is able to discover *logical lists* in a varying set of websites (that is, it is not domain dependent). Moreover, the quality of the results are not correlated to how the lists are rendered in web pages (i.e. horizontal, vertical and tiled). In average, it achieves 100% for Precision, 95% for Recall and a F-Measure 97%. With respect to the ground truth, the algorithm does not extract any False Positive, and it outputs only 466 False Negatives. In general, it returns perfect results (100% precision and recall) for several kind of websites spanning different applications domain, but there are some of them which presents values for recall ranging from 81% and 91%. Considering "last.fm", which gave a recall equal to 81%, I found that the presentation of the data records is sometime quite different, because of the high variance in the number of the "similar to" tags (which are presented as HTML <a>) assigned to each listing. Analyzing other examples such as "IlSole24Ore.it" and "RealEstateSource.au" we found the same problem, that is, the presentation of the data records is quite variable across the web pages, and

so the HyLiEn algorithm sometimes misses some of the data records. Anyway we see that the proposed algorithms is effective is able to discover *logical lists* on different type of websites.

Website	*Ground*	*TP*	*FN*	*Precision*	*Recall*	*F-measure*
BariToday.it	904	904	0	100%	100%	100%
Subito.it	1000	1000	0	100%	100%	100%
GitHub.com	100	100	0	100%	100%	100%
TestoLegge.it	360	360	0	100%	100%	100%
Zoopla.co.uk	597	597	0	100%	100%	100%
FindAProperty.co.uk	60	60	0	100%	100%	100%
Savills.co.uk	232	232	0	100%	100%	100%
AutoTrader.co.uk	60	60	0	100%	100%	100%
EbayMotors.com	3925	3925	0	100%	100%	100%
Doogal.co.uk	38240	38240	0	100%	100%	100%
RealEstateSource.com	368	316	62	100%	85%	91%
AutoWeb.co.uk	180	180	0	100%	100%	100%
TechCrunch.com	434	422	12	100%	95%	98%
Landsend.com	1243	1243	0	100%	100%	100%
TMZ.com	300	300	0	100%	100%	100%
IlSole24Ore.it	510	445	65	100%	81%	86%
GoBari.it	350	340	10	100%	97%	98%
AGI.it	60	60	0	100%	100%	100%
BBCNews.co.uk	347	310	37	100%	89%	94%
milano.corriere.it	30	30	0	100%	100%	100%
torino.repubblica.it	70	68	2	100%	98%	99%
Ansa.it	1506	1479	27	100%	98%	99%
LeMonde.fr	445	418	27	100%	94%	97%
Time.com	377	377	0	100%	100%	100%
aur.ArchLinux.org	575	575	0	100%	100%	100%
Immobiliare.it	609	536	73	100%	86%	93%
bitbucket.org	130	130	0	100%	100%	100%
MyMovies.com	563	515	48	100%	92%	96%
Trulia.com	3300	3300	0	100%	100%	100%
YouTube.com	580	567	13	100%	98%	99%
FileStube.com	332	304	28	100%	91%	95%
Last.fm	60	41	19	100%	68%	81%
Bing.com	130	130	0	100%	100%	100%
addons.mozilla.org	984	939	45	100%	95%	97%
AutoScout24.com	840	840	0	100%	100%	100%
Facebook.com	2820	2820	0	100%	100%	100%
SlideShare.net	2037	2037	0	100%	100%	100%
Gazzetta.it	970	970	0	100%	100%	100%
ElPais.es	294	285	9	100%	98%	99%
StackOverflow	585	585	0	100%	100%	100%
Sums and Averages	66.527	66.061	466	100%	95%	97%

Table 2.1: Discovered *logical list* elements for websites dataset.

3

Exploiting Web Sites Structural and Content Features for Web Pages Clustering

The process of automatically organizing web pages and websites has always attracted extensive attention in several research areas because of its significant theoretical challenges and because of its great application and commercial potentials. Clustering represents one of the most investigated techniques used to evaluate the interaction among web pages and organize them into groups such that web pages within the same group are more similar to each other than those belonging to different ones. As a consequence, clustering algorithms can be profitably used to organize the content of websites and to understand their hidden structure or, in Information Retrieval, to provide an effective representation of search engine results [ZE98].

Since a web page is characterized by several representations (i.e. textual representation, structural representation based on HTML tags and visual features and structural representation based on hyperlinks) the existing clustering algorithms differ in their ability of using these representations.

In particular, clustering based on textual representation typically group web pages using words distribution [CAC08, HGKI02, ZE98]. Solutions based on this approach manage web pages as plain text corpora ignoring all the other information of which is enriched. These algorithms 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 (e.g. web pages related to professors, courses, books, lists of publications of a single researchers) but characterized by different distribution of terms.

On the other side, clustering based on structure typically considers the HTML formatting (i.e. HTML tags and visual information rendered by a web browser) [CMM05, But04, BG10]. Algorithms, which use these information to organize web pages, are based on idea that web pages are automatically generated by programs that extract data from a back-end database and embed them into an HTML template. Web pages generated with this approach, show a common structure and layout, while differing in content. However, because tags are more often used for styling purposes than for content structuring, it can happen that most web pages in a website have the same structure, even if they refer to distinct semantic types. The effect is that the clustering algorithm will generate very few low-quality clusters.

These two solutions, however, only exploit within-page information. On the contrary, other solutions also exploit the graph defined by the hyperlink structure of a set of web pages [LZW$^+$10, QD06]. Clustering on hyperlink structure is based on idea that web pages, differently from traditional textual documents, are enriched by hyperlinks that enable information to be split in multiple and interdependent web pages. These hyperlinks can be used to identify collections of web pages semantically related and relationships among these collections. For example, the website of a computer science department may organize its web pages in collections of courses, research areas, and people; people may be further split into faculty, staff and students. Analyzing the link structure allow us to discover the hidden structure of website, which can be used to aid the navigation and the understanding of a website. However the effectiveness of clustering algorithms strongly depends by links density and presence of noisy links.

Although there are several works that focus on web pages clustering, most of them analyze contents, web page structure (i.e. HTML formatting) and hyperlink structure of a website almost independently, mainly because different sources of information use different data representations. Over the last decade, some researchers tried to combine different sources of information. For example, [HZHDDS02, MS00, WK02, DBS05, AS06, LZW$^+$10] combine content and hyperlink structure for web page clustering, [GTL$^+$02, HNVV03, CCM$^+$03, QD06, ZYCG07] combine content and hyperlink structure for classification and

[CMM05] combines web page and hyperlink structure for clustering purposes.

This chapter is intended to be a contribution in the direction of performing clustering of web pages in a website by combining information about content, web page structure and hyperlink structure of web pages. This goal is achieved analyzing web pages' HTML formatting to extract from each page collections of links, called *web lists*, which can be used generate a compact and noise-free representation of the website's graph. Then, the extracted hyperlink structure and content information of web pages are mapped in a single vector space representation which can be used by any traditional and best-performing clustering algorithms.

Specifically, the proposed approach is based on the idea that two web pages are similar if they have common terms (i.e. *Bag of words hypothesis* [TP10]) and they share the same reachability properties in the website's graph. In order to consider reachability, the solution I propose is inspired by the concept of Distributional hypothesis, initially defined for words in natural language processing (i.e. "*You shall know a word by the company it keeps*")[Fir57] and recently extended to generic objects [GGV15]. In the context of the Web it is possible to translate that citation in "*You shall know a web page by the paths it keeps*". According to this hypothesis two web pages are similar if they are involved in the same paths in the website's graph.

This chapter is organized as follows: In the next section related work on clustering of web pages are described. In Section 3.2 the proposed solution is detailed. Results empirically evaluated in Section 3.3.

3.1 Related Work and Motivations

Many techniques have been used for clustering large corpora of web pages. In particular, existing algorithms are based on two main approaches: *content analysis* and *structure analysis*.

As clarified before, the most important clustering algorithms based on **web pages contents** treat web pages as independent textual documents. This is the case of [AWP14, CAC08, HGKI02, ZE98], where the words distribution is used to discover appropriate groups of interrelated web pages. The advantage of this approach is that many off-the-shelf clustering tools based on vector space model can be directly applied. However, these approaches fail to learn accurate models due the uncontrolled and heterogeneous nature of web page contents. In fact, traditional clustering algorithms are based on the assumption that textual documents share consistent writing styles, provide enough contex-

tual information, are plain and completely unstructured, and are independent and identically distributed. These limitations are more obvious for clustering of web pages belonging to different websites or whose content is created in a collaborative manner. In this case, in fact, web pages related to the same topic could be contextually different, contain similar information into web elements with different semantic rules (e.g., navigational menu, main content, calendar, tables, and logotypes) and different functionalities (e.g., links, buttons, images, anchor-texts, etc.) [QD09].

On the contrary of plain text documents, web pages are characterized by structural properties such as HTML tags, visual features and hyperlinks which define the structural representations of web pages. It as been proved (see [ZYCG07, CMM05, BG10, LZW$^+$10]) that such structural information provide different and complementary information respect to a textual representation.

A different perspective is represented by clustering algorithms that exploit structural properties. They can be organized in two main categories: *i)* clustering based on HTML formatting and *ii)* clustering based on hyperlink structure. Clustering based on **HTML formatting** takes advantage of the structural and visual information embedded in HTML tags, which are usually ignored by plain text approaches. In the literature several works (see, for instance [But04, HAB12]) propose to organize web pages according with their structural information. They are mainly based on the assumption that HTML documents can be treated as XML documents. However, differently from XML tags, HTML tags are oriented towards data visualization rather than data representation. The consequence is that it can appear that web pages of the same semantic type (e.g., web pages of professors) are codified and rendered using different tags structures or, viceversa, different types of web pages are codified with same tags structure. For example, structured data can be coded with a HTML table (<table>) tag or HTML list (<ul>) and have similar appearance. This defaces the quality of generated clusters.

To overcome this limit, in [BG10, CMM05], the authors proposed to use visual information associated with HTML tags. Specifically, in [BG10] clustering is based only on visual properties of web pages. Goal of this approach is to group web pages that could not have similar content, could not share the same HTML structure, but just look visually similar. In [CMM05] layout and visual properties associated with HTML tags are used to characterize the structure of the whole web page, and collections of hyperlinks in a web page are used to find pages with similar structural representation.

Although visual information and HTML tags can improve the quality of

web page clustering, algorithms based on HTML formatting work well for well-structured and homogeneous web pages. This limits their usage to web pages belonging to the same website and automatically generated from a back-end database (e.g., Deep Web Databases) or generated from dedicated content management systems (CMSs).

Clustering algorithms based on the **hyperlink structure** work on a inter-related collection of web pages. The basic idea is that when two web pages are connected via a link, there is some semantic relationship between them, which can be the basis for the partitioning of the collection into clusters. In general, such methods (see [CCdM+03]) only use direct links among web pages and, on the base of them, they use/define some similarity measures (e.g., bibliographic coupling [Kes63], co-citation [Sma73], etc.). However, as claimed in [FE03], algorithms based on the hyperlink structure work well when the hyperlink structure is dense and noise-free. In fact, clustering based on the hyperlink structure returns low quality results for web pages without sufficient amount of in-links or out-links. Moreover, not all the links are equally important for the clustering process: web pages are often enriched with noisy hyperlinks such as hyperlinks used to enforce the web page authority in a link-based ranking scenario or with short-cut hyperlinks. To overcome these issues several algorithms combine content information with link information (see [HZHDDS02, MS00, WK02, DBS05, AS06, LZW+10]).

In this context, the earliest methods (for web page clustering) that try to combine textual information with co-citation and bibliographic coupling measures are [MS00, WK02, DBS05]. Additionally, [HZHDDS02] faced the problem of noisy links, considering only hyperlinks among topically similar web pages and co-cited web pages. In particular, it associates a weight which combines the content similarity and co-citation to each edge (i, j) connecting the web pages i and j in the website graph. Afterwards, a traditional clustering algorithm based on graph partitioning is adopted. The method has two main limitations: i) textual information are used only to weight links, then two web pages sharing same distributional properties but having low textual similarity cannot be clustered together; ii) the graph clustering algorithm is NP-hard. In [AS06] a relaxation-labeling-based algorithm which first clusters documents based on their contents and then re-assigns the computed labels using the hyperlink structure among immediate neighbors is developed. To avoid the potential increase in the level of noise, only a subset of good neighbors are considered.

An alternative approach to better organize web pages belonging to the same website and remove noisy links is to filter link collections having similar vi-

sual and/or structural properties [CMM05, QD06, WJH13, LZW+10, LFC+14]. However, in my knowledge only [CMM05] and [LZW+10] use information in link collections to improve clustering results. In [LZW+10] 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 appear more time together in a link collection. However, each measure needs a different representation space and their combinations is an open issue.

Previous solutions consider only direct relationships among neighbors, without analyzing the global structure of the website graph. [ZCY10, GGV15, TQW+15, PARS14] claim that the similarity between two graph's nodes can be expressed in terms of similarity of their respective contexts, that is, how they share surrounding nodes (which could not necessary immediately neighbors). In [ZCY10], the authors propose a graph clustering algorithm that focuses on topological structure of a network and node properties, which can be textual or relational. A set of attribute nodes and attribute edges are added to the original graph. With such graph augmentation, the attribute similarity is transformed to vertex vicinity in the graph: two vertices which share an attribute value are connected by a common attribute vertex. Then, a neighborhood random walk model, which measures the vertex closeness on the augmented graph through both structure edges and attribute edges, unifies the two similarities. Although the algorithm is able to combine structural and content information using a common representation, it cannot be applied to data having numerical values (e.g. tf-idf) or categorical attributes with a huge amount of distinct values.

In [PARS14, TQW+15] two embedding methods are proposed (DeepWalk and Line, respectively). They exploit neural networks to generate a low-dimensional and dense vector representation of graph's nodes. DeepWalk [PARS14] applies the skip-gram method on truncated random walks to encode long-range influences among graph's nodes. However, this approach is not able to capture the local graph structure (i.e. nodes which can be considered similar because are strongly connected). Line [TQW+15] optimizes an objective function that incorporates both the local (i.e. direct neighbors) and global (i.e. neighbors of neighbors) network structures. However, while DeepWalk is able to consider influences of arbitrary length, Line is able only to capture influence of length two. Moreover, both methods ignore node attributes (e.g. textual content). Consequently, clustering based on generated embedding might be difficult in graphs without sufficient in-links or out-links, but characterized by rich textual contents.

3.2 Methodology

As anticipated, the problem I consider is that of clustering web pages belonging to the same website by combining content, web page structure and hyperlink structure in a vector space representation. To achieve this goal, the proposed solution implements a four steps strategy: in the first step website crawling is performed. Crawling uses web pages' structure information and exploits web lists in order to mitigate problems coming from noisy links. The output of this phase is the website graph, where each node represents a single page and edges represent hyperlinks. In the second step, I generate a link vector by exploiting Random Walks extracted from the website's graph. In the third phase content vectors are generated. In the last phase, a unified representation of pages is generated and clustering is performed on such representation.

3.2.1 Website crawling

A Website can be formally described as a direct 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 representing the website's entry page allows the website to be viewed as a rooted directed graph.

As claimed in [CMM05] 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 clustering process, such as hyperlinks used to enforce the web page authority in a link-based ranking scenario, short-cut hyperlinks, etc. . Moreover, the website structure is codified in navigational systems which provide a local view of the website organization. Navigational systems (e.g. menus, navbars, product lists) are implemented as hyperlink collections having same domain name and sharing layout and presentation properties. In this respect, the solution I propose, based on the usage of web lists, has a twofold effect: from one side it guarantees that only urls useful to the clustering process 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 web pages [CMM05, QD06, WJH13, LFC$^+$14]. The crawling algorithm is described in Algorithm 3. In particular, starting from the homepage h, the method *extractWebLists()* (see Algorithm 1) is iteratively applied to extract url collections having same domain of h and organized in *web lists* (see Def. 5). Only web pages included in web lists are further explored.

Fig.3.1 shows, in red boxes, web lists extracted from the homepage of a computer science department which will be used to guide the website crawler.

Figure 3.1: In red rectangles the web lists extracted from a web page taken from *www.cs.illinois.edu* are highlighted

Algorithm 3 crawlingWebsite(homepage)

Input: URL homepage;

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

```
1: frontier = Set()
2: Q = Queue(homepage)
3: repeat
4:    currentPage = Q. dequeue();
5:    text = currentPage. getText();
6:    V. add((currentPage, text));
7:    webLists =  extractWebLists(currentPage);
8:    for each list ∈ webLists do
9:       pagesToAnalyze = list. filterNot(page → frontier. contains(page) );
10:      Q. enqueue(pagesToAnalyze);
11:      frontier. add(pagesToAnalyze);
12:      for each u ∈ pagesToAnalyze do
13:         E. add((currentPage, u));
14:      end for
15:   end for
16: until !queue.empty() return (V, E)
```

Links in box A will be excluded because their domains are different from the homepage's domain.

To identify from a web page the set of web lists Algorithm 1 is used (see Section 2.3.1). Since our goal is to extract the navigation systems encoded in a web page which are charachterized by a simple and a very similar HTML structure, the normalized edit distance is used here to evaluate the structural similarity among web elements.

The output of website crawling step is the sub-graph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$, which will be used for link and content vectors generation steps.

3.2.2 Link vectors generation through Random Walks

A random walk over a linked structure is based on the idea that the connections between nodes encode information about their correlations. Then, the effective semantic of any node in a graph is obtained by analyzing how it is correlated to all other nodes. To capture and codify correlations among graph's nodes (i.e. web pages), which can be indirect, the Random Walk with Restart (RWR)

Algorithm 4 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. getRandomVertex();
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
```

approach is used.

RWR is a Markov chain describing the sequence of nodes (i.e. web pages) visited by a random walker: starting from a random point i, with probability $(1-\alpha)$ a walker stochastically walks to a new, connected neighbor node or, with probability α, it restarts his walk from i. Algorithm 5 describes the generation process.

In order to use the topology of the graph for capturing correlations among nodes, the generated random walks must be sufficiently long to encode those information. However, on the other side, it is advisable to avoid the effect described in [PL05], that is, when the length of a random walk starting at vertex i tends towards infinity, the probability of being on a vertex j does not depend on the starting vertex i. In our solution, the length of the random walk, $rwrLength$, is defined by the user (following indications provided in [PL05], in the experiments, I set $rwrLength = 10$).

Inspired by the information retrieval universe, a web page can be seen as a word, that is, a topic indicator and each random walk as a document constituting the natural context of words (i.e. topical unity). Then, continuing the information-retrieval metaphor, we can represent a collection of random walks as a document collection where topics intertwine and overlap. The idea is to apply any NLP or information retrieval algorithm which uses the distributional hypothesis on document's objects to extract new knowledge [Sah08].

For this scope, I apply a state-of-art algorithm, the skip-gram model [MSC+13] to extract a vector space representations of web pages that encode the topological structure of the website. In the skip-gram model we are given a word w in a corpus of words V_W (in our case a web page w belonging to random walks) and their contexts $c \in V_C$ (in our case web pages in random walks which appear before and after the web page w).

We consider the conditional probabilities $p(c|w)$, and given a random walks collection Rws generated by Algorithm 5, the goal is to set the parameters θ of $p(c|w;\theta)$ so to maximize the following probability:

$$\underset{\theta}{\operatorname{argmax}} \prod_{L \in Rws; w \in L} \left[\prod_{c \in C_L(w)} prox_L(w,c) \cdot p(c|w;\theta) \right] \tag{3.1}$$

where L is a random walk in Rws, w is a web page in L and $C_L(w) = \{w_{i-k}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+k}\}$ is the set of contexts of web page w in the list L. Moreover, $prox_L(w,c)$ represents the proximity between w and $c \in C_L(w)$. This is necessary since the skip-gram model gives more importance to the nearby context words than distant context words.

One approach for parameterizing the skip-gram model follows the neural-network language models literature, and models the conditional probability $p(c|w;\theta)$ using soft-max:

$$p(c|w;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in V_C} e^{v_{c'} \cdot v_w}} \tag{3.2}$$

where v_c, $v_{c'}$ and $v_w \in \mathbb{R}^d$ are vector representations for c, c' and w respectively (d is defined by the user). Therefore, the optimization problem (3.1) leads to the identification of the web page and context matrices $W = \{v_{w_i} | w_i \in V_W\}$ and $C = \{v_{c_i} | c_i \in V_C\}$. They are dependent each other and we only use W to represent web pages (coherently with what proposed in [MSC+13] for words).

Equation 3.2 is computationally expensive due the summation $\sum_{c' \in V_C}$. One way of making the computation more tractable is to replace the softmax with an *hierarchical softmax*. It uses a binary tree representation where words (web pages in our case) are leaves and each node stores the relative probabilities of its child nodes. Using this representation it is possible to evaluate only $\sim log_2(V_C)$ nodes rather than V_C [MSC+13].

An alternative approach to hierarchical softmax is represented by the soft-max negative-sampling approach (SGNS). In this case the objective function can be formalized as follows:

$$\underset{\theta}{\operatorname{argmax}} \prod_{(w,c) \in D} p((w,c) \in D | c, w; \theta) \prod_{(w,c) \notin D'} p((w,c) \in D | c, w; \theta) \tag{3.3}$$

where:

- D is the set of all web pages and context pairs which are extracted from the website;
- D' is the set of random pairs, assuming they are not present in the web site (i.e. negative examples). In [MSC+13] authors suggest a number of negative examples of 5-20 pairs (for each w) for small corpus data and 2-5 pairs (for each w) for big corpus data;
- $p((w,c) \in D|c,w;\theta) = \frac{1}{1+e^{-v_c \cdot v_w}}$ is the probability that the pair (w,c) comes from the corpus data;
- $p((w,c) \notin D|c,w;\theta) = 1 - p(D = 1|c,w;\theta)$ is the probability that the pair (w,c) does not come from the corpus data;

Therefore, given in input to skip-gram model a corpus data composed by the collection of random walks, it returns the matrix W which embeds each web page into a dense and low-dimensional space $\mathbb{R}^d$.

3.2.3 Content vectors generation

In this section I describe the process for generating a vector representation of web pages using textual information. On the contrary of from traditional documents, web pages are written in HTML and contain additional information, such as HTML tags, hyperlinks and anchor text or other than textual content visible in a web browser. To apply on web pages a bag-of-words representation we need to compute a preprocessing step, in which the following operations are performed:

- Remove HTML tags. However, I maintain terms in anchor, title and metadata since they contribute to better organize web pages [FAN04]
- Unescape escaped characters;
- Eliminate non-alphanumeric characters;
- Eliminate too frequent ($> 90\%$) and infrequent ($< 5\%$) words;

After preprocessing, each web page is converted in a plain textual document and we can apply the traditional *TF-IDF* weighting schema to obtain a content-vector representation. Due the uncontrolled and heterogeneous nature of web page contents, vector representation of web pages based on content is characterized by high-dimensional sparse data. To obtain a dense and low-dimensional

space the Truncated SVD algorithm is applied. It is a low-rank matrix approximation based on random sampling [HMT11]. In particular, given the *TF-IDF matrix* of size $|V'| \times n$ and the desired dimensionality of content vectors m, where $m << n$, the algorithm returns a denser and lower-dimensional matrix of size $|V'| \times m$.

3.2.4 Content-link coupled Clustering

Once the content vector $v_c \in \mathbb{R}^m$ and the link vector $v_l \in \mathbb{R}^d$ of each web page in V' have been generated, the last step of the algorithm is to concatenate them in a new vector having dimension $m+d$. Before the concatenation step I normalize each vector with its Euclidean norm. In this way we ensure that components of v_l having highest weights are as important as components of v_c having highest weights.

The matrix generated by concatenation step preserves both structural and textual information and can be used in traditional clustering algorithms based on vector space model. In this study I consider K-MEANS [Jai10] and H-DBSCAN [CMS13] because they are well known and present several complementary properties (distance vs. density-based).

3.3 Experiments

In order to empirically evaluate the proposed approach, I performed experiments on several real websites. Specifically, I used four computer science department's websites: *Illinois* (cs.illinois.edu), *Princeton* (cs.princeton.edu), *Oxford* (www.cs.ox.ac.ou), *Stanford* (cs.stanford.edu). The motivation behind this choice is related to our competence in manually labelling pages belonging to this domain. This was necessary in order to create a ground truth for the evaluation of the clustering results.

This evaluation has been performed in order to answer to specific research questions: 1) Which is the real contribution of combining content and hyperlink structure in a single vector space representation with respect to using only either textual content or hyperlink structure? 2) Which is the real contribution of exploiting web pages structure (i.e. HTML formatting) and, specifically, the role of using web lists to reduce noise and improve clustering results?

In Table 3.1 the dimension of each dataset is described. In particular, to correctly analyze the contribution of web lists in the clustering process, I compare only the web pages extracted both by crawling websites using web lists and by traditional crawling (first column of Table 3.1). Moreover, I report the

Table 3.1: Description of Websites

Website	#pages	#edges	#edges using web lists	#clusters
Illinois	563	9415	5330	10
Oxford	3480	44526	35148	19
Stanford	167	12372	30087	10
Princeton	3132	122493	104585	16

dimension of the edge set obtained with traditional crawling (second column) and crawling using web lists (third column). Finally the last column describes the number of clusters manually identified.

I evaluated the effectiveness of the proposed approach by using the following measures:

- Homogeneity [RH07]: each cluster should contain only data points that are members of a single class. This measure is computed by calculating the conditional entropy of the class distribution given the proposed clustering. It is bounded below by 0.0 and above by 1.0 (higher is better).

- Completeness [RH07]: all of the data points that are members of a given class should be elements of the same cluster. Symmetrically to Homogeneity it is computed by the conditional entropy of the proposed cluster distribution given the real class. It ranges between 0 and 1 (higher is better).

- V-Measure [RH07]: harmonic mean between homogeneity and completeness.

- Adjusted Mutual Information (AMI): it is a variation of the Mutual Information MI. $MI = \sum_{i \in K} \sum_{j \in C} log \frac{P(i,j)}{P(i)P(j))}$ where C is the set of real classes, K is the set of learned clusters, $P(i,j)$ denotes the probability that a point belongs to both the real class i and the learned cluster j and $P(i)$ is the a priori probability that a point falls into i. However MI is generally higher for two clusterings with a larger number of clusters, regardless of whether there is actually more information shared. The Adjusted Mutual Information represents an adjustment of this metric to overcome this limitation.

- Adjusted Random Index (ARI) [HA85]: it represents a similarity measure between two clusterings by considering all pairs of samples and counting pairs that are assigned in the same or different clusters in the predicted

and true clusterings. $RI = (a+b)/\binom{n}{2}$ where a is number of pairs of points that are in the same class and learned cluster and b is number of pairs of points that belong to different class and learned cluster. As in the case of AMI, the ARI is an adjustment which ensures to have a value close to 0.0 for random labeling independently of the number of clusters and samples and exactly 1.0 when the clusterings are identical.

- Silhouette: it measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters.

To response to the previous research questions I ran the proposed algorithm with different configurations depending on the crawling type (with or without web list constrains) and web pages' information used for the vectors generation. I enumerate the used configurations as follows:

- *Text*. I generate a vector space representation, having dimension $m = 120$, using only web pages' textual information;
- *RW-List*. I generate a vector space representation of size $d = 120$ using only hyperlink structure extracted by crawling the website using web lists. For this scope I ran the *rwrGeneration* algorithm (see Section 3.2.2) setting the input parameters to $\alpha = 1$, $rwrLength = 10$ and $dbLength = 100k$;
- *RW-NoList*. I generate a vector space representation of size 120 using only the hyperlink structure obtained with traditional crawling. I ran *rwrGeneration* with the same parameters of RW-List;
- *Comb-Lists*. I combine, as defined in Section 3.2.4, the content vector of size $m = 60$ and hyperlink structure vector of size $d = 60$ generated by crawling the website using web lists. For link vector generation I use the same parameters of other configurations.
- *Comb-NoLists*. As in the Comb-Lists configuration I combine textual and hyperlink structure of web pages in a single vector space representation having size 120. However, on the contrary of Comb-Lists, I use traditional crawling.

Since the goal is not that of comparing clustering algorithms, I set for K-MEANS the parameter K (i.e. total number of clusters to generate) to the number of real clusters, while I set for H-DBSCAN the *minimal cluster size* parameter to 5. Finally, since at the best of my knowledge there is no work which uses the

skip-gram model to analyze the topological structure of websites, I ran both of skip-gram versions (i.e hierarchical softmax and SGNS) for generating link vectors. However, since experiments do no show relevant defferences between the two approaches, I report only results for SGNS (setting the window size to 5).

Tables 3.2, 3.3, 3.4 and 3.5 present the main results. In general, the experiments show that best results are obtained combining textual information with hyperlink structure. This is more evident for Illinois and Oxford websites, where content and hyperlinks structure codify complementary information for clustering purpose. However, for the Stanford website using the textual information decreases the clustering performance. The importance of combining content and hyperlink structure is confirmed by Nemenyi post-hoc test (see Figure 3.2) and Wilcoxon signed Rank test (see Table 3.6). This behaviour is quite uniform for all the evaluation measures considered (see Table 3.6).

For the last research question, results do not show a statistical contribution in the use of web lists for clustering purpose (see Figure 3.2 and Table 3.6). This can be motivated by the fact that analyzed websites are very well structured and poor of noisy links. This can be observed in Table 3.1, where there is not a big difference in terms of edges number between the real web graph and that one extracted using web lists. However, as expected the Completeness is higher for Comb-Lists, confirming that clusters have higher "precision" in the case of crawling based on web lists (see Figure 3.2 b).

Table 3.2: Experimental result for Illinois's website

Configuation	Clustering	Homogeneity	Completeness	V-Measure	ARI	AMI	Silhouette
Text	KMEANS	0.84	0.62	0.71	0.4	0.61	0.33
Text	H-DBSCAN	0.72	0.53	0.61	0.4	0.5	0.21
RW-Lists	KMEANS	0.72	0.53	0.61	0.27	0.51	0.42
RW-Lists	H-DBSCAN	0.81	0.47	0.6	0.18	0.43	**0.43**
RW-NoLists	KMEANS	0.71	0.52	0.6	0.25	0.5	0.42
RW-NoLists	H-DBSCAN	0.8	0.45	0.58	0.17	0.41	0.42
Comb-Lists	KMEANS	**0.9**	**0.69**	**0.78**	**0.54**	**0.68**	0.4
Comb-Lists	H-DBSCAN	0.83	0.51	0.63	0.27	0.48	0.34
Comb-NoLists	KMEANS	0.84	0.62	0.71	0.37	0.6	0.38
Comb-NoLists	H-DBSCAN	0.83	0.52	0.64	0.27	0.49	0.29

Table 3.3: Experimental results for the Princeton's website

Configuation	Clustering	Homogeneity	Completeness	V-Measure	ARI	AMI	Silhouette
Text	KMEANS	0.71	**0.59**	**0.64**	**0.68**	**0.58**	0.21
Text	H-DBSCAN	0.36	0.31	0.34	0.12	0.28	-0.21
RW-Lists	KMEANS	0.56	0.37	0.45	0.27	0.36	0.18
RW-Lists	H-DBSCAN	0.49	0.3	0.37	0.12	0.26	-0.05
RW-NoLists	KMEANS	0.55	0.36	0.43	0.24	0.35	0.15
RW-NoLists	H-DBSCAN	0.48	0.3	0.37	0.1	0.26	-0.09
Comb-Lists	KMEANS	0.76	0.54	0.63	0.55	0.53	0.14
Comb-Lists	H-DBSCAN	0.47	0.52	0.49	0.36	0.45	**0.37**
Comb-NoLists	KMEANS	**0.78**	0.54	**0.64**	0.49	0.53	0.13
Comb-NoLists	H-DBSCAN	0.47	0.52	0.49	0.37	0.45	0.38

Table 3.4: Experimental results for the Oxford's website

Configuation	Clustering	Homogeneity	Completeness	V-Measure	ARI	AMI	Silhouette
Text	KMEANS	0.74	0.6	0.66	0.48	0.59	0.25
Text	H-DBSCAN	0.43	0.41	0.42	0.07	0.37	-0.06
RW-Lists	KMEANS	0.65	0.55	0.6	0.48	0.54	0.32
RW-Lists	H-DBSCAN	0.6	0.44	0.51	0.26	0.41	0.22
RW-NoLists	KMEANS	0.67	0.57	0.62	0.51	0.56	**0.35**
RW-NoLists	H-DBSCAN	0.6	0.45	0.51	0.27	0.41	0.18
Comb-Lists	KMEANS	0.79	0.67	0.73	**0.56**	0.67	0.34
Comb-Lists	H-DBSCAN	0.58	0.49	0.53	0.15	0.47	0.08
Comb-NoLists	KMEANS	**0.81**	**0.68**	**0.74**	0.53	**0.68**	0.28
Comb-NoLists	H-DBSCAN	0.62	0.53	0.57	0.23	0.51	0.08

Table 3.5: Experimental results for the Stanford's website

Configuation	Clustering	Homogeneity	Completeness	V-Measure	ARI	AMI	Silhouette
Text	KMEANS	0.37	0.43	0.39	0.08	0.28	0.3
Text	H-DBSCAN	0.18	0.62	0.28	0.07	0.16	0.43
RW-Lists	KMEANS	**0.59**	0.58	**0.58**	**0.27**	**0.52**	0.31
RW-Lists	H-DBSCAN	0.28	0.4	0.33	0.1	0.22	0.15
RW-NoLists	KMEANS	0.47	0.54	0.5	0.14	0.39	0.53
RW-NoLists	H-DBSCAN	0.34	0.6	0.43	0.13	0.29	**0.55**
Comb-Lists	KMEANS	0.42	0.46	0.44	0.12	0.34	0.22
Comb-Lists	H-DBSCAN	0.21	**0.63**	0.31	0.07	0.17	0.46
Comb-NoLists	KMEANS	0.53	0.56	0.54	0.17	0.46	0.35
Comb-NoLists	H-DBSCAN	0.34	0.51	0.4	0.12	0.28	0.27

Table 3.6: Wilcoxon pairwise signed Rank tests. (+) indicates that the second model wins. (-) indicates that the first model wins. The results are highlighted in bold if the difference is statistically significant (at p-value=0.05). The tests have been performed by considering the results obtained with both hierarchical softmax and SGNS skip-gram models.

	Homogeneity	Completeness	V-Measure	Adj Rand index	Adj Mutual info	Silhouette
Text vs Comb	(+) **0.000**	(-) 0.055	(+) **0.000**	(+) 0.342	(+) **0.003**	(+) **0.020**
RW vs Comb	(+) **0.002**	(+) **0.000**	(+) **0.000**	(+) **0.000**	(+) **0.000**	(+) 0.229
NoLists vs Lists	(-) 0.342	(-) 0.970	(-) 0.418	(+) 0.659	(+) 0.358	(-) 0.362

(a) Homogeneity

(b) Completeness

(c) V-Measure

(d) AMI

4

Automatic Generation of Sitemaps Based on Navigation Systems

One of the recognized problems in the web mining community 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 [NL06]. 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 [Fan07]. 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 [LLC05, OC03]. 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

[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.) [CMM05].

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 [KMH13], 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) [LYHL10].

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 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 [ADW01].

Several works face the problem of the automatic extraction or generation of sitemaps (also called hierarchies or taxonomies). They are usually based on

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

analysis of text, hyperlinks, urls structure, heuristics or a combination of these features [LNL04, YL09, LCC11, WBH12]. 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 chapter 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*")[Fir57] and recently extended to generic objects [GGV15]. 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. transform discovered patterns in order 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.

4.1 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, I discuss related works in these three research fiedls.

4.1.1 Sitemap Extraction

The problem of generating website hierarchies is part of a more the general research topic of Web Mining. Although extracting website's sitemaps is an important task for many web applications, few studies specifically focus on this problem. In the following I revise some related works done in web mining research area, even if not directly related to the specific task of sitemaps generation. I 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 [AZ12] 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 of web structure mining algorithms is that if there is a hyperlink between two pages, then some semantic relation may exist between them [CMM05]. 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 [LCC11], 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 [KN12] 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 [LCC11], 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 [LCC11] 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 [YL09] 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 [WBH12] 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.

4.1.2 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 [BBA$^+$00, Mob07]. 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* [Mob07]. 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 [NM03] 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.

4.1.3 Automatic extraction of web lists

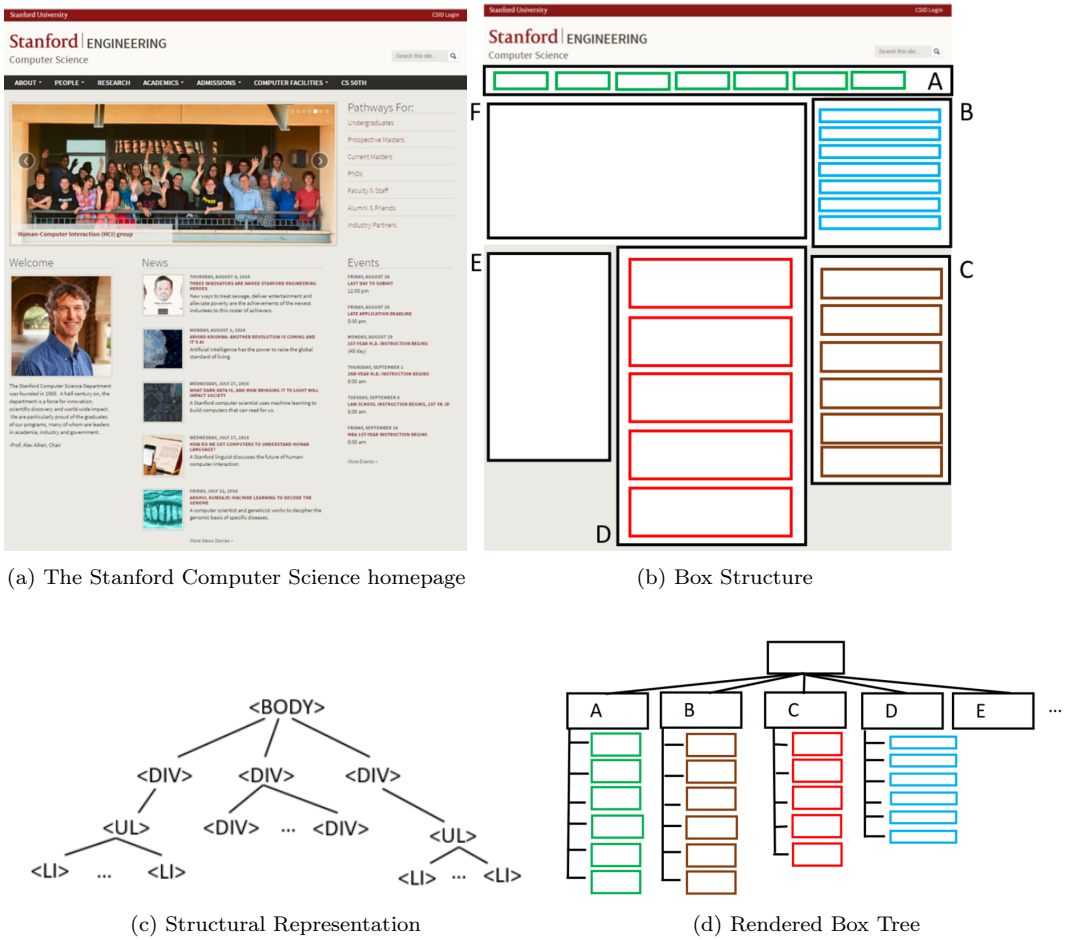
As claimed in [CMM05], not all the links are equally important to describe the website structure. Several works in the field of Web Mining exploit web pages taking the advantage of the structural and visual information embedded in HTML tags. In [CMM05, LYHL10, LFC+14] collections of hyperlinks having 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, [CMM05, LYHL10] identify web lists and exploit them for the task of web page clustering. Specifically, in [CMM05] 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 [LYHL10] 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 [LFC+14] 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.

I follow this basic idea: using 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.

4.2 Extraction of Sitemaps: Preliminary Definitions

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



(a) The Stanford Computer Science homepage

(b) Box Structure

(c) Structural Representation

(d) Rendered Box Tree

Figure 4.1

As detailed in Section 2.2, a web page is characterized by multiple representations: *i)* a textual representation composed by web page terms; *ii)* a visual representation, composed by its rendered box tree (see Definition 4); *iii)* a structural representation, composed by HTML tags (see Definition 2). Figure 4.1 shows an example of multiple representations of a web page.

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

Definition 10 *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, the 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 11 ***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 I will refer to the relative support, unless otherwise specified:

$$\sigma(S) = \frac{|\{t \in SDB : S \text{ is a sub-sequence of } t\}|}{|SDB|} \tag{4.1}$$

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.

Following the suggestion provided in [Mob07], we also exploit the definition of contiguous sequence which is formally defined as follows:

Definition 12 ***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 sub-sequences. 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 13 ***Closed Sequence****: A sequential pattern (contiguous sequential pattern) S_j is* ***closed*** *if and only if it is frequent and its support is different from all of its frequent super-sequences.*

Example 2 *Figure 4.3(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 14 ***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. 5);*
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 14. 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.

4.3 Methodology

The problem we consider is that extracting the website's sitemap, defined according to Definition 14. To achieve this goal, I combine different techniques

which have roots in web and data mining research areas: *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. Details of this step are described in Section 3.2.1. 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's graph. In the third phase we mine closed frequent sequences of urls (i.e., closed frequent navigation paths) in form of a tree. In the last step the tree of closed sequences is analyzed in order to extract the sitemap. This step requires the transformation of the tree of sequences in a tree of web pages such that the obtained tree has the homepage as 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.

4.3.1 Sequence Database Generation

To capture and codify correlations among graph's nodes (i.e. web pages) I 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 [PPLM06], 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 [PPLM06] we generate short random

Algorithm 5 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
```

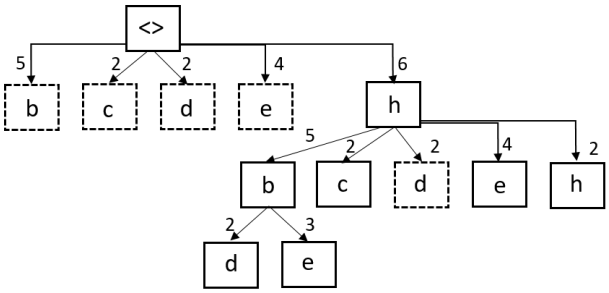
walks in the way that the random walker tends to get "trapped" into densely connected parts of the graph corresponding to communities (i.e. sibling pages in the sitemap) and for avoiding to infer false correlations among web pages due the nodes' degree. Algorithm 5 describes the generation process of the sequence dataset. Figure 4.3b shows a sequence database obtained from the web graph in Figure 4.3a

4.3.2 Closed sequential patter mining

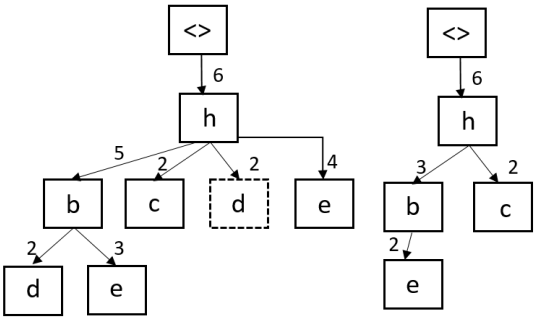
Given the sequence database (SDB), extracted in the Section 4.3.1, and a user defined threshold t (i.e. minimum support), we apply a closed sequential pattern mining algorithm to extract all the frequent sequences. In this phase we used the algorithm CloFAST [FLCM16] as closed sequential pattern mining 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 $\sigma : E' \to \mathbb{R}$ 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 4.2a shows an example of tree extracted by CloFAST for the input sequence dataset in figure 4.3b.

Since a contribution of this chapter is to compare the sitemaps extracted considering two different types of sequences (i.e. sequential patterns and con-



(a)



(b) (c)

Figure 4.2: (a) Frequent sequence tree extracted by CloFAST with absolute minsup = 3 and SDB in fig. 4.3(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.

tiguous sequential pattern), I extend CloFAST by allowing it to also extract contiuous sequential patterns.

To extract the weighted tree of contiguous sequences from $T_{CloFAST}$ I 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:

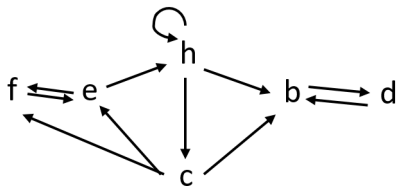
Definition 15 (Vertical Id-list) *Let SDB be a sequence database of size* n *(i.e. $|SDB| = n$), $S_j \in SDB$ its j-th sequence ($j \in \{1, 2, \ldots, n\}$), and α a sequence associated to a node of the tree, its* vertical id-list*, denoted as VIL_α, is a vector of size n, such that for each $j = 1, \ldots, n$*

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

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

Example 3 *Figure 4.3c shows the VIL of the sequence $\alpha = \langle h, b \rangle$. Values in VIL_α represent the end position of the occurrences of the sequence α in the sequences of Figure 4.3b. 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 h (i.e. b is the last item in α), in the first sequence S_1. The second element is (list with values 2 and 4) the position of the first item b (after a) in the sequence S_2. The other values are respectively list with only value 3 (for sequences S_3), list with only value 2 (for S_4)* null *(for S_5) and list with only value 3 (for S_6). In particular, the fifth element is null since α is not present in S_5. Finally, 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 6) 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 $T'_{CloFAST}$



(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]
[2]
[3]
NULL
[3]

(c)

FIGURE 4.3: (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 walk with restart from the homepage; (c) VIL for the sequence $\alpha = \langle h, b \rangle$.

4.3.3 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 4.2b the web page e can be reached using the frequent patterns $h \rightarrow b \rightarrow e$ and $h \rightarrow e$. 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

$$\sum_{e' \in T_\beta} w(e') + w(\beta) > \sum_{e \in T_\alpha} w(e) + w(\alpha) \tag{4.2}$$

The first constraint ensures that frequent paths that do not exist in the crawled graph G are removed (see Def. 14.3). 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

Algorithm 6 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) return vil;
```

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. 14, 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 6.

4.4 Experiments and Discussion

The aim of this chapter is to answer to the following research questions: 1) Which is the real contribution of combining the random walk teory with structured data with respect to use the content and the whole hyperlink structure?

2) Which is the real contribution of extracting hierarchies based on sequential patterns with respect to extract hierarchies based on *contiguous* sequential patterns. For this reason we compare the performances of SMAP (i.e. sitemap generation through sequential patterns) and $SMAP_{sc}$ (i.e. sitemap generation through *contiguous* sequential patterns) with a state-of-art algorithm, that is HTDM [WBH12].

The empirical evaluation of these algorithms was not a trivial task because, at the best of our knowledge, there is no dataset generated for the specific task of sitemap extraction. Thus, two solutions were possible: *i)* involving human experts to extract the hierarchical organization of websites and generating a ground truth dataset for each considered website; *ii)* using existing sitemap pages, which are manually generated by web masters and provided by some website, as ground truth.

In the first case, the involvement of experts requires extensive human effort and working time. This is not feasible in the context of the Web especially for websites having a great amount of web pages strongly connected and websites having deep hierarchies. In the second case, sitemap pages are in general manually created by web designers. Therefore, 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 by few pages). For this reason, to empirically evaluate SMAP, I have performed experiments on following websites which provide updated sitemap pages: *cs.illinois.edu*, *www.cs.ox.ac.uk*, *www.cs.princeton.edu*, and *www.enel.it*.

The evaluation has been performed in terms of Precision, Recall and F-measure of edges. In particular, the Precision measures how many of the extracted edges belong to the real sitemap. The Recall measures how many edges, belonging to the real sitemap are extracted. I also included the F-Measure which is the weighted harmonic means of Precision and Recall. These measures are evaluated counting how many edges of the real sitemap are found by the algorithm under analysis (i.e. SMAP, $SMAP_{cs}$, HDTM). More formally:

$$Precision = \frac{|\{e|e \in SMAP(G) \wedge e \in GT\_Sitemap\}|}{|\{e|e \in SMAP(G)\}|} \tag{4.3}$$

$$Recall = \frac{|\{e|e \in SMAP(G) \wedge e \in GT\_Sitemap\}|}{|\{e|e \in GT\_Sitemap\}|} \tag{4.4}$$

$$F = \frac{2(precision \times recall)}{precision + recall} \tag{4.5}$$

In these formulas $SMAP(G)$ represents the set of edges extracted by $SMAP$,

whereas $GT\_Sitemap$ represents the set of edges in the real sitemap (ground truth).

Table 4.1 presents the main results. For HDTM I set $\gamma = 0.25$ (to avoid too shallow or too deep hierarchies) and $5,000$ Gibbs iterations, as suggested by the authors in their paper. For SMAP and $SMAP_{cs}$ I set $rwrLength = 10$ and $dbLength = 500.000$ (See Section 4.3.1). Moreover, I analyze the effectiveness of SMAP and $SMAP_{cs}$ varying the minimum support. It is interesting to note that when the minimum support threshold decreases, we are able to obtain deeper and wider sitemaps. While, 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. This behaviour can be observed by analyzing the F-measure which, from website to website, shows different trends for different values of minimum support.

Interestingly, both versions of our algorithm significantly outperform HDTM, independently of the website and of the minimum support threshold. This can be motivated by the different nature of the two algorithms: on the contrary of our approach, HDTM organizes website's pages in a hierarchy using the distribution of web pages' terms. Then, it can 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 in different parts of the extracted hierarchy.

Finally comparing the sitemaps extracted using sequential patterns and *contiguous* sequential patterns we can observe that in general SMAP outperforms $SMAP_{sc}$. This is due by the fact that $SMAP_{sc}$ extracts narrower and shallower hierarchies compared with $SMAP$. In fact, setting low support thresholds $SMAP_{sc}$ obtain better results than SMAP in terms of Recall but, since it is unable to extract the deepest levels, it obtains lower results in terms of Precision.

Website	Algorithm	min. supp.	Precision	Recall	F-Measure
cs.illinois.edu	SMAP	0.005	0.38	**0.78**	0.51
cs.illinois.edu	$SMAP_{cs}$	0.005	0.1	0.65	0.17
cs.illinois.edu	SMAP	0.001	0.66	0.48	**0.56**
cs.illinois.edu	$SMAP_{cs}$	0.001	0.35	0.76	0.48
cs.illinois.edu	SMAP	0.0005	0.66	0.33	0.44
cs.illinois.edu	$SMAP_{cs}$	0.0005	0.35	0.33	0.34
cs.illinois.edu	SMAP	0.0001	**0.75**	0.26	0.38
cs.illinois.edu	$SMAP_{cs}$	0.0001	0.61	0.35	0.45
cs.illinois.edu	HDTM	–	0.12	0.1	0.11
cs.ox.ac.uk	SMAP	0.005	**0.72**	0.3	**0.42**
cs.ox.ac.uk	$SMAP_{cs}$	0.005	0.27	**0.41**	0.33
cs.ox.ac.uk	SMAP	0.001	**0.72**	0.21	0.33
cs.ox.ac.uk	$SMAP_{cs}$	0.001	0.60	0.18	1.28
cs.ox.ac.uk	SMAP	0.0005	0.72	0.21	0.33
cs.ox.ac.uk	$SMAP_{cs}$	0.0005	0.60	0.17	0.27
cs.ox.ac.uk	SMAP	0.0001	**0.72**	0.21	0.33
cs.ox.ac.uk	$SMAP_{cs}$	0.0001	0.60	0.17	0.27
cs.ox.ac.uk	HDTM	–	0.37	0.15	0.21
cs.princeton.edu	SMAP	0.005	0.61	**0.55**	**0.58**
cs.princeton.edu	$SMAP_{cs}$	0.005	0.1	0.22	0.14
cs.princeton.edu	SMAP	0.001	0.89	0.23	0.36
cs.princeton.edu	$SMAP_{cs}$	0.001	0.6	0.55	0.58
cs.princeton.edu	SMAP	0.0005	0.89	0.2	0.33
cs.princeton.edu	$SMAP_{cs}$	0.0005	0.6	0.32	0.41
cs.princeton.edu	SMAP	0.0001	**0.91**	0.13	0.23
cs.princeton.edu	$SMAP_{cs}$	0.0001	0.73	0.18	0.29
cs.princeton.edu	HDTM	–	0.36	0.08	0.13
enel.it	SMAP	0.005	0.31	0.74	0.43
enel.it	$SMAP_{cs}$	0.005	0.1	0.47	0.16
enel.it	SMAP	0.001	0.36	0.72	0.48
enel.it	$SMAP_{cs}$	0.001	0.31	0.73	0.43
enel.it	SMAP	0.0005	0.8	0.73	0.76
enel.it	$SMAP_{cs}$	0.0005	0.31	0.73	0.43
enel.it	SMAP	0.0001	**0.8**	0.73	0.76
enel.it	$SMAP_{cs}$	0.0001	0.77	**0.8**	**0.79**
enel.it	HDTM	–	0.35	0.75	0.48

Table 4.1: Experimental results of SMAP, $SMAP_{cs}$ and HDTM.

5

CloFAST: Closed Sequential Pattern Mining using Sparse and Vertical Id-Lists

5.1 Introduction

Since its introduction [AS95], sequential pattern mining has become a fundamental data mining task with a large spectrum of applications, including web mining [JZ10], classification [ETPF08], finding copy-paste bugs in large-scale software code [LLMZ06] and mining motifs from biological sequences [TLS+09].

In sequential pattern mining, input data is a set of sequences, called *data-sequences*. Each data-sequence is an ordered list of *transactions*, where each transaction is a set of literals, called *itemset*. Typically, the order of transactions in the list is based on the time-stamp associated with each transaction, although other non time-related orderings are possible. The output of sequential pattern mining is sequential patterns, each of which consists of a list of items. The problem is to find all sequential patterns with a user-specified minimum *support* (or *frequency*), which is defined as the percentage of data-sequences that contain the pattern.

If compared with the more common problem of frequent pattern mining, sequential pattern mining is computationally challenging because, when solving this problem, a combinatorially explosive number of intermediate subsequences has to be generated and/or tested [Han05]. In fact, although algorithms presented in the literature are relatively efficient [PHMA+01, AFGY02, Zak01, SHJ05, YK05], when they are used to mine long sequences, time and space scal-

ability becomes increasingly critical. This is especially true for low values of the support threshold.

To alleviate this problem, research in sequential pattern mining has made progress in two directions: *i)* efficient methods for mining only the set of closed sequential patterns and *ii)* efficient methods for pruning the search space and exploiting specifically designed data structures.

As for *i)*, many studies pinpoint the idea that for mining frequent sequential patterns, one should not mine *all* the frequent sequences [YHA03, WHL07, MPT09, FM10]. In particular, they propose mining the *closed* sequential patterns, where a sequential pattern α is closed if it has no proper supersequence β with the same support. Intuitively, since all the subsequences of a frequent sequence are also frequent, mining the set of closed sequential patterns may help avoid the generation of unnecessary subsequences, thus leading to more compact results and saving computational time and space costs.

As for *ii)*, many algorithms avoid maintaining the set of already generated closed sequences during the mining process [YHA03]. Pruning of the search space and closure checking typically exploit multiple pseudo-projected databases [WHL07] (i.e. databases of sequences generated from a single sequence prefix), which are designed to be efficiently queried. However, pseudo-projected databases require significant time and space to be created and queried, thus limiting not only the capability of the algorithms to mine large datasets with long data-sequences, but also the capability of the algorithm to process dense data-sequences (i.e. data-sequences whose itemsets contain many items).

Several approaches (e.g. ClaSP [GCMG13] and SPADE [Zak01]) attempt to overcome the limits of pseudo-projected databases by exploiting a vertical representation formalism. However, they all start with 1-itemset sequences and extend them by iteratively alternating *sequence extension*, i.e. appending an itemset to a sequence, and *itemset extension*, i.e. adding an item to an itemset in the sequence. In this way, a frequent itemset mining step is required at each iteration, with a computational cost that does not scale well with the size of frequent sequences.

In this chapter I propose CloFAST (**Clo**sed **FAST** sequence mining algorithm based on sparse id-lists), a novel algorithm to mine closed sequences from large databases of long sequences. It extends and revises the algorithm FAST [SFMH11] that extracts only frequent sequences. In particular, CloFAST, similarly to FAST, combines a new data representation of the dataset (*sparse id-list* and *vertical id-list* [SFMH11]) to fast count the support of sequential patterns. However, differently from FAST, it exploits the properties of *sparse id-lists* and

of *vertical id-lists*, in order to define a novel one-step technique for *sequence closure checking* and *search space pruning*. Similarly to BIDE [WHL07], CloFAST, during the mining process, does not need to maintain the set of already mined closed sequences [YHA03] to prune the search space and to check if newly discovered frequent sequential patterns are closed.

CloFAST does not build pseudo-projected databases and does not need to scan them. The initial dataset of sequences of transactions is read once for all to create both *sparse id-lists* and *vertical id-lists*, which are two distinct indexes loaded in the main memory. Sparse id-lists store the position of the transactions which contain a given itemset, while vertical id-lists store the position of a given sequential pattern in the input sequences.

CloFAST uses sparse id-lists to mine closed frequent itemsets and to enumerate the search space, while it uses vertical id-lists to generate the closed sequence patterns. The support of itemsets and sequences is efficiently computed from the sparse id-lists and the vertical id-lists, without requiring additional database scans. Moreover, in order to check the (non-)closure of a considered sequential pattern α and to consequently prune the search space, I propose a novel technique, called *backward closure checking*, which checks whether a new sequence pattern β, obtained by adding a new item/itemset at any position (not necessarily at the end) in α, has the same support as α. In this case, α cannot be considered closed.

Finally, CloFAST mines closed frequent itemsets only at the beginning of the mining process, in order to obtain an initial set of sequences. New sequences are then generated by directly working on the sequences, without generating frequent itemsets.

The contributions of this chapter are the following:

1. I propose a two-step process that performs *(i)* closed itemset mining, and *(ii)* closed sequential pattern discovery. The two steps only work on sparse id-lists and vertical id-lists, thus gaining efficiency both in time and space.

2. I study formal properties of sparse id-lists and vertical id-lists, which can be used for closed sequential pattern mining.

3. I propose an efficient *backward closure checking* which works on sparse id-lists and vertical id-lists.

4. I present a new *pruning method*, performed during the backward closure checking, which removes non-promising enumerations during the generation of closed sequential patterns.

5. I theoretically prove the correctness and completeness of closed sequential patterns generated by both CloFAST with the backward closure checking technique and CloFAST with pruning.

6. I present empirical evidence that CloFAST outperforms competing algorithms on several real-world and artificially generated sequence datasets.

5.2 Problem Definition and Background

Let us consider a sequence database *SDB* of customer transactions. In particular, a sequence represents the (ordered) list of transactions associated to a customer and each transaction consists of a set of items purchased. Each sequence is uniquely identified by a sequence identifier (*sequence-id* or *SID*), while each transaction in the sequence is uniquely identified by a transaction identifier (*transaction-id* or *TID*). The *size* of *SDB* ($|SDB|$) corresponds to the number of sequences (i.e. the number of customers) in the sequence database. In Table 5.1, I report an example of SDB with three sequences (i.e. $|SDB| = 3$): the first sequence contains five transactions, the second sequence contains two transactions, while the third sequence contains three transactions.

More formally, let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of distinct items, which can be sorted according to some lexicographic ordering $\leq_l$ (e.g. alphabetic ordering). A customer sequence S is a list of transactions, $S = \langle t_1, t_2, \ldots, t_m \rangle$, where each $t_j \subseteq I$ denotes the set of items bought in the j-th transaction. The *size* $|\alpha|$ of a sequence α is the number of itemsets (transactions) in the sequence. A sequence $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ is a **subsequence** of a sequence $\beta = \langle b_1, b_2, \ldots, b_n \rangle$, if and only if integers $i_1, i_2, \ldots, i_m$ exist, such that $1 \leq i_1 < i_2 < \ldots < i_m \leq n$ and $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_m \subseteq b_{i_m}$. We say that β is a **supersequence** of α or that β *contains* α.

Example 4 *The sequence* $\beta = \langle \{a, b\}, \{c\}, \{d, e\} \rangle$ *is a supersequence of* $\alpha = \langle \{a\}, \{d\} \rangle$ *because* $\{a\}$ *is a subset of* $\{a, b\}$ *and* $\{d\}$ *is a subset of* $\{d, e\}$*. On the contrary,* β *is not a supersequence of* $\lambda = \langle \{c, d\} \rangle$*, since the itemset* $\{c, d\}$ *is not contained in any itemset of* β*.*

Given a sequence β, its **absolute support** in SDB is the number of sequences in *SDB* which contain β, while its **relative support** is the absolute support divided by $|SDB|$. Henceforth, $\beta : s$ will denote the sequence β and its absolute support s, and the term support will refer to the absolute support, unless otherwise specified.

SID	Sequence
1	$\langle\{a,b,f\},\{d\},\{e\},\{a\},\{d\}\rangle$
2	$\langle\{e\},\{a\}\rangle$
3	$\langle\{e\},\{a,b,f\},\{b,d,e\}\rangle$

Table 5.1: An example of a sequence database (SDB)

Given two sequences β and α, if β is a supersequence of α and their absolute (or relative) support in SDB is the same, we say that β **absorbs** α. A sequential pattern α is **closed** if no proper sequence β that absorbs α exists.

The problem of closed sequence mining is formulated as follows:
Given a sequence database SDB and a minimum support threshold min_sup, find all the closed sequential patterns in SDB, such that their support in SDB is at least min_sup. Generated patterns are called **closed frequent** sequential patterns.

Example 5 *Table 5.1 shows an example of a sequence database. If min_sup* $= 2$*, the complete set of closed frequent sequences consists of only four sequences:* $\langle\{a,b,f\},\{d\}\rangle : 2$*,* $\langle\{a,b,f\},\{e\}\rangle : 2$*,* $\langle\{e\},\{a\}\rangle : 3$*,* $\langle\{e\},\{a\},\{d\}\rangle : 2$*,*
while the total number of frequent sequences is 26.

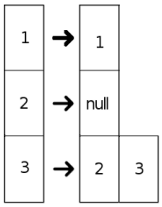
The algorithm proposed in this work uses two data structures, called *sparse id-list* (SIL) and *vertical id-list* (VIL), recently introduced in [SFMH11] for frequent sequence mining. They are an optimized representation of the database, since their size is bound by the size of the input dataset. The concept of *id-list* was first introduced by SPADE [AFGY02], where an *id-list* of a sequence α was defined as the list of all input customer-id and transaction-id pairs containing α in the database. In the following, I formally introduce them.

Let SDB be a sequence database of size n (i.e. $|SDB| = n$) and $S_j \in SDB$ the j-th customer sequence ($j \in \{1, 2, \ldots, n\}$).

Definition 16 ***Sparse id-list****: Given an itemset $t \subseteq 2^I$, its* sparse id-list*, denoted as SIL_t, is a vector of size n, such that for each $j = 1, \ldots, n$*

$$SIL_t[j] = \begin{cases} \textit{the list of the ordered transaction-ids of } t \textit{ in } S_j & \textit{if } S_j \textit{ contains } t \\ \textit{null} & \textit{otherwise} \end{cases}$$

Example 6 *Fig. 5.1(a) shows the SIL_a and $SIL_{a,b}$ of the itemsets $\{a\}$ and $\{a,b\}$ respectively. The values represent the position of the relative itemset in the database in Table 5.1. Other examples of SILs for the same database are reported in Fig. 5.2(a) and (b).*



SID	Sequence
1	$\langle\{a,b,f\},\{d\},\{e\},\{a\},\{d\}\rangle$
2	$\langle\{e\},\{a\}\rangle$
3	$\langle\{e\},\{a,b,f\},\{b,d,e\}\rangle$

(c)

Figure 5.1: From left to right: (a) the sparse id-lists for itemset $\{b\}$, (b) the sparse id-lists for itemset $\{a,b\}$, (c) the database of sequences.

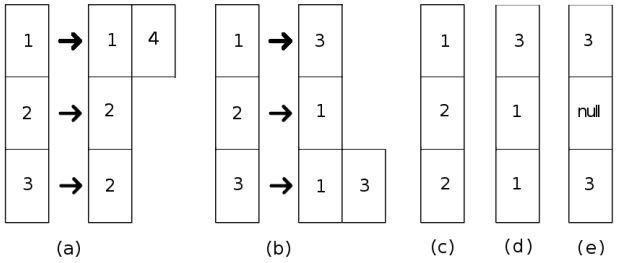


Figure 5.2: (a) sparse id-list for the itemset $\{a\}$; (b) sparse id-list for the itemset $\{e\}$; (c) vertical id-list for the sequence $\langle\{a\}\rangle$; (d) vertical id-list for the sequence $\langle\{e\}\rangle$; (e) vertical id-list for the sequence $\langle\{a\},\{e\}\rangle$.

Definition 17 ***Vertical id-list****: Given a sequence α, whose last itemset is i, its* vertical id-list*, denoted as VIL_α, is a vector of size n, such that for each $j = 1, \ldots, n$*

$$VIL_\alpha[j] = \begin{cases} \textit{the transaction-id of } i \textit{ in the first occurrence of } \alpha \textit{ in } S_j & \textit{if } S_j \textit{ contains } \alpha \\ \textit{null} & \textit{otherwise} \end{cases}$$

Example 7 *Fig. 5.2(c), (d) and (e) show some VILs. In particular Fig. 5.2(e) shows the VIL_α of the sequence $\alpha = \langle\{a\},\{e\}\rangle$. Values in VIL_α represent the ending position of the first occurrence of the sequence α in the sequences S_j of Table 5.1. In particular, the first element (value 3) represents the position of the first occurrence of $\{e\}$, after $\{a\}$ ($\{e\}$ is the last itemset in α), in the first sequence. The second element is null since α is not present in the second sequence. The third element (value 3) represents the position of the first occurrence of $\{e\}$ (after $\{a\}$) in the third sequence.*

5.3 Related Work

To the best of our knowledge, CloSpan [YHA03], BIDE [WHL07], ClaSP [GCMG13] and COBRA [HCTH06] represent the state of the art in closed sequential pattern mining. CloSpan is based on the *candidate maintenance and test approach*, which generates a candidate set for closed sequential patterns, enumerates the search space and then performs post-pruning. It uses the *equivalence of projected databases* to stop the search and prune the search space. The basic idea is that if a sequence β is a supersequence of a discovered sequence α and the number of items in the corresponding projected databases is the same, then the projected databases are equal and it is possible to stop the search of any descendant of α, since both α and β have the same support.

Wang et al. [WHL07] proposed BIDE as an alternative solution which has the advantage of avoiding candidate maintenance. They presented the *BI-Directional Extension* schema to generate closed sequences and *BackScan* to prune the search space. The BI-Directional Extension schema is based on the idea that a sequence $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ is not closed if an item/itemset a' exists such that it can be used to extend α to a new sequence β, having the same support as α. In particular, β can be obtained from α through either a *forward-extension* (adding a new item/itemset after a_m) or a *backward-extension* (adding a new item/itemset before a_j, with $1 \leq j \leq m$). If no such item/itemset exists, then α is closed. BIDE does not keep track of any candidate closed sequential patterns for sequence closure checking. This means that it needs multiple scans of the projected databases for both the bi-directional closure checking and the BackScan pruning.

Both CloSpan and BIDE adopt the PrefixSpan [PHMA+01] approach in the mining phase. PrefixSpan is a pattern-growth divide-and-conquer algorithm that grows sequences by itemset extension and sequence extension. In particular, PrefixSpan grows a prefix pattern to obtain longer sequential patterns by building and scanning its projected database. Although frequent sequences in the projected databases are enumerated to reduce computational complexity, its time complexity is strictly related to the size of the projected databases. For databases with long sequences and large transactions, discovering the local frequent itemsets for each projected database could become an expensive process.

These limitations have been overcome by both SPADE [Zak01] and SPAM [AFGY02], which work on more efficient data structures. Improvements are obtained by using a vertical database/bitmap representation (*id-lists*) of the database for

both itemsets and sequences. In this way, both itemset extension and sequence extension steps are executed by joining/ANDing operations between vertical/bitmap representation of sequence candidates. Experimental results presented in [AFGY02, GCMG13] show that both SPAM and SPADE outperform PrefixSpan on *large* datasets, because they avoid the Prefixspan cost for local frequent itemset mining.

The approach used by SPADE has been recently extended in ClaSP [GCMG13] for closed sequential pattern mining. In particular, ClaSP exploits the concept of a vertical database format to obtain closed sequences without making several scans of the input database. According to the authors, this significantly improves performances over existing algorithms such as CloSpan. Drawing inspiration from this observation, I decided to exploit both sparse and vertical id-lists (SILs and VILs) to fast count the support of sequential patterns in CloFAST. Contrary to SPADE and ClaSP, where the large size of the id-lists negatively affects the computational time of the joins, in CloFAST both the itemset extension and the sequence extension are based on SILs and VILs, which can be efficiently used in support counting, sequence closure checking, and search space pruning (see Section 5.6) without performing temporal joins.

Note that all previously referenced algorithms 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.

To explain the first drawback (redundant itemset extension) I report a simple example. Consider a database of two sequences:

$SDB = [\ \langle\ \{a,b\},\ \{a,b,c\},\ \{a,b\}\ \rangle,\ \langle\ \{a,b,c\},\{a,b\},\ \{a,b\}\rangle]$.

In this case, finding the closed sequence $\langle\ \{a,b\},\ \{a,b\},\ \{a,b\}\rangle$ generally requires three item extensions of $\{a\}$ with $\{b\}$ and three sequence extensions which add $\{a\}$ to the sequence. Graphically, the following steps are typically necessary:

$\mapsto \langle\ \{a\}\ \rangle \rightarrow \langle\ \{a,b\}\ \rangle \mapsto \langle\ \{a,b\},\ \{a\}\ \rangle \rightarrow \langle\ \{a,b\},\ \{a,b\}\ \rangle \mapsto \langle\ \{a,b\},\ \{a,b\},\ \{a\}\ \rangle$
$\rightarrow \langle\ \{a,b\},\ \{a,b\},\ \{a,b\}\ \rangle$

where $\rightarrow$ indicates the itemset extension and $\mapsto$ indicates the sequence extension. However, if we discover that item $\{a\}$ is not closed (since $\{a,b\}$ absorbs $\{a\}$), then we can directly perform *sequence* extensions of $\{a,b\}$, instead of generating *item* extensions of $\{a\}$. This means that only the following operations are necessary:

$\mapsto \langle\ \{a,b\}\ \rangle \mapsto \langle\ \{a,b\},\ \{a,b\}\ \rangle \mapsto \langle\ \{a,b\},\ \{a,b\},\ \{a,b\}\ \rangle$

Obviously, this requires a preliminary closed frequent itemset mining step.

The second drawback (expensive matching cost) is due to queries on (previously generated) projected databases, in order to obtain, after pattern-growth, new projected databases. This process in not trivial since we are working on databases of sequences and a query means a complete scan of the previously generated projected database. Moreover, it is noteworthy that both itemset extension and sequence extension require the generation of a new projected database.

COBRA attempts to overcome these two drawbacks. Instead of extending a pattern by iteratively alternating *i)* itemset extension and *ii)* sequence extension, it separates the two phases and generates closed frequent itemsets before mining closed sequential patterns. Sequences are extended by only performing sequence extension. Therefore, the closed sequence mining is composed of three consecutive phases: *i)* search for all closed frequent itemsets; *ii)* transformation of the original dataset into a horizontal format (similar to projected databases); *iii)* enumeration of closed sequential patterns.

It is noteworthy that this approach is not equivalent to mining all closed frequent itemsets, then encoding different itemsets as different symbols and finally applying any (non-closed) sequence pattern mining algorithm (*à la* AprioriAll [AS95], for sequential pattern mining). Indeed, the notions of supersequence/subsequence used to identify closed sequences are based on the notions of superset/subset of itemsets, which cannot be evaluated after encoding. Consequently, the enumeration of closed sequential patterns cannot be based only on input closed itemsets, but it requires additional information extracted during the phase of mining closed itemsets.

CloFAST follows the same approach as COBRA. The difference is that COBRA generates all the sequences of the same length and then performs an expensive post-pruning (called *ExtPruning*) to discard non-closed sequences, while CloFAST applies an *on-line* (i.e. during the sequence generation phase) pruning strategy which operates on vertical id-lists. Moreover, the computation of the pattern support in COBRA requires the identification of the first occurrence of the itemset in each sequence, while in CloFAST it is performed by simply counting the non-null elements in the *vertical id-list* of the pattern. This means that COBRA has to analyze sequences, whereas CloFAST does not.

5.4 The Closed Itemset Enumeration Tree and the Closed Sequence Enumeration Tree

In this section I present the two main data structures used in CloFAST, that is, the Closed Itemset Enumeration Tree (CIET) and the Closed Sequence Enumeration Tree (CSET). The former is used to store closed frequent itemsets, while the latter is used to store the closed frequent sequential patterns. Similar to the *Lexicographic Sequence Tree* introduced in CloSpan [YHA03], we assume that a lexicographic ordering $\leq_l$ exists in the set of items I. This ordering, as explained in [YHA03], can be extended for sequences composed of itemsets, by exploiting the concepts of sub/superset and sub/supersequence (see Section 5.2). For the sake of simplicity, I will use the same notation $\leq_l$ for this extension of the ordering.

5.4.1 Closed Itemset Enumeration Tree (CIET).

Similar to a *Set Enumeration Tree* [ZDR00], the CIET is an in-memory data structure that allows us to enumerate the complete set of closed frequent itemsets. It is characterized by the following properties: 1) each node in the tree corresponds to an itemset and the root is the empty itemset ($\emptyset$); 2) if a node corresponds to an itemset i, its children are obtained by itemset extensions from i; 3) the left sibling of a node precedes the right sibling in the lexicographic order (see Figure 5.3 for an example).

Formally, this tree structure is defined as follows:

- the root node of the tree is labeled with $\emptyset$;
- the first level enumerates the frequent 1-item itemsets (i.e. itemsets with a single item in I) according to the ordering $\leq_l$;
- for other levels, nodes represent frequent k-item itemsets, with $k > 1$. Each node is constructed by merging the itemset of its parent node with the itemset of a sibling of its parent node.

Only nodes for (candidate) closed itemsets are added to the CIET. Inspired by the classification of the nodes in Moment [CWYM06], we label each node in the CIET as:

- *intermediate*: the node represents a subset of a closed itemset represented in one of its descendant nodes;

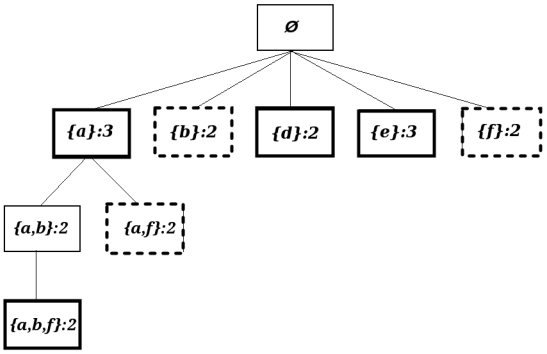


Figure 5.3: The CIET for our running example. Nodes with thick borders represent closed itemsets. Nodes with dashed borders represent unpromising nodes. The remaining nodes represent intermediate nodes.

- *unpromising*: the node represents a subset of a closed itemset represented in other branches of the tree;
- *closed*: a node is labeled as closed if it represents a closed itemset.

Figure 5.3 shows an example of a CIET for the database in Table 5.1, when $min\_sup = 2$. Each node contains a frequent itemset and its corresponding support. CloFAST traverses the CIET in a depth-first search order. Only the descendants of the nodes labeled as *closed* or *intermediate* are explored. Indeed, descendants of an unpromising node can be pruned since they cannot represent additional closed itemsets. To check whether or not a certain node corresponding to an itemset i should be labeled as *unpromising*, CloFAST needs to know whether there is a frequent itemset j, such that j absorbs i but does not descend from i. For this purpose, a hashmap (i.e. a structure that maps keys to values) is used to store the set of the closed frequent itemsets associated to a support value, which represents the key of the hashmap. It is noteworthy that nodes labeled as closed can be changed to intermediate during the tree construction.

5.4.2 Closed Sequence Enumeration Tree (CSET).

The mined set of closed itemsets is used in the construction of the CSET, which enumerates the complete search space of closed sequences, similarly to the se-

quence tree described in [SFMH11]. For the CSET it is possible to define the following properties: 1) each node in the tree corresponds to a sequence and the root corresponds to the *null* sequence (Λ); 2) if a node corresponds to a sequence s, its children are obtained by a sequence extension of s.

This tree has the following structure:

- the root node of the tree is labeled with Λ;
- nodes at the first level represent *candidate* **closed sequences of size 1**, whose unique element is either *i)* a closed frequent itemset corresponding to a node labeled as closed in the CIET or *ii)* an itemset labeled as intermediate in the CIET for which its SIL is different from the SIL of its closed descendant node;
- nodes at higher first levels represent **sequences of size greater than 1**. Each node can be constructed in two ways: *i)* by adding to the sequence of its parent node u the last itemset of the sequence in a sibling of u; *ii)* by adding to the sequence of its parent node u the last itemset of the sequence in u itself. The latter guarantees that sequences containing multiple repeated occurrences of the same item/itemset are not discarded (e.g. $\langle\{a,b,f\},\{a,b,f\}\rangle$ in Example 8). In any case, only nodes for frequent and (candidate) closed sequences are added to the tree.

According to the previous definition, two sibling nodes of a CSET correspond to two distinct sequences of itemsets, $\alpha = \langle a_1, a_2, \ldots, a_m\rangle$ and $\beta = \langle b_1, b_2, \ldots, b_m\rangle$, such that $a_m \neq b_m$ and $\forall i = 1, \ldots, m-1 :\ a_i = b_i$.

Each node in the closed sequence enumeration tree can be labeled as: (i) *closed*, (ii) *non-closed* and (iii) *pruned*.

Figure 5.4 shows an example of CSET for the database in Table 5.1 with $min\_sup = 2$. Each node in the figure contains a frequent sequence and its corresponding support. Different borders (thick, dashed or plain) are used for different labeled nodes.

CloFAST builds the CSET in a depth-first search order. Each node in the CSET is considered for sequence-extension. In order to exemplify how nodes at the second and at subsequent levels are constructed, I report a simple example:

Example 8 *Consider the sequence extension of node 2 in Figure 5.4. In this case, the candidate sequences are:* $\langle\ \{a,b,f\},\ \{d\}\ \rangle$, $\langle\ \{a,b,f\},\ \{a\}\ \rangle$, $\langle\ \{a,b,f\},\ \{e\}\ \rangle$, $\langle\ \{a,b,f\},\ \{a,b,f\}\ \rangle$. *Obviously, not all of them are frequent sequences and are added to the CSET.*

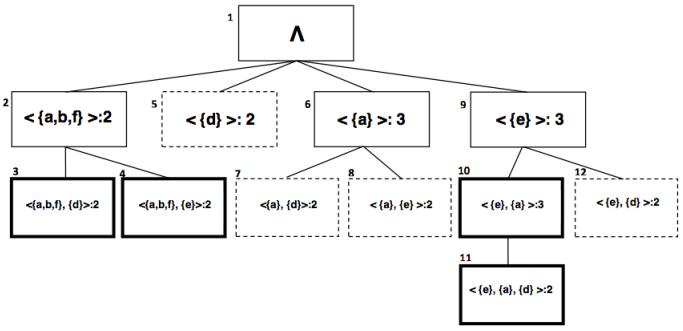


Figure 5.4: The CSET for our example. Nodes with thick borders represent (candidate) closed sequences. Nodes with dashed borders represent pruned nodes. Remaining nodes represent non-closed sequences.

5.5 Properties of SILs and VILs for efficient mining of closed sequential patterns

In this section I present several properties of VIL and SIL data structures which can be profitably exploited by the sequential pattern mining algorithm.

Proposition 1 *Let* $\alpha = \langle a_1, \ldots, a_m \rangle$*, such that* $VIL_\alpha[j] \neq null$*. Then, for each i=1, ..., m* $- 1$*,* $VIL_{\langle a_1,\ldots,a_i \rangle}[j] < VIL_{\langle a_1,\ldots,a_i,a_{i+1} \rangle}[j]$*.*

Proof 1 *It follows from VIL definition.*

Proposition 2 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$*,* $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$*,* $VIL_{\alpha\epsilon}[j] \neq null$*. Then,* $VIL_\alpha[j] \neq null$*.*

Proof 2 *It follows from the VIL definition.*

These two propositions express two necessary conditions on the VIL structure, when the j-th sequence in SDB contains α or the composed sequence $\alpha\epsilon$.

Proposition 3 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$*,* $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$*,* $VIL_{\alpha\epsilon}[j] \neq null$*,* γ *any sequence. If* $VIL_\gamma[j] = VIL_\alpha[j]$*, then* $VIL_{\gamma\epsilon}[j] \neq null$*.*

Proof 3 *Let* $p = VIL_\gamma[j] = VIL_\alpha[j]$*,* $\langle b_1, \ldots, b_p, b_{p+1}, \ldots, b_r \rangle$*,* $r \geq m$*, be the j-th subsequence in SDB. From the VIL definition, it follows that the subsequence* $\langle b_1, \ldots, b_p \rangle$ *contains both* α *and* γ*. Moreover, the subsequence* $\langle b_{p+1}, \ldots, b_r \rangle$ *contains* ϵ*. Therefore, the j-th sequence in SDB also contains* $\gamma\epsilon$*, i.e.* $VIL_{\gamma\epsilon}[j] \neq null$*.*

This proposition expresses an important property related to the containment of composed sequences. If the j-th sequence in SDB contains α, $\alpha\epsilon$ and γ, and the position of the last itemset in α coincides with the position of the last itemset in γ, then the j-th sequence also contains $\gamma\epsilon$.

Proposition 4 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$, $VIL_\alpha[j] \neq null$, $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, $VIL_\gamma[j] \neq null$, $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i \rangle$. *If* $VIL_\gamma[j] < VIL_\alpha[j]$, *then* $VIL_\beta[j] = VIL_\alpha[j]$.

Proof 4 *It is obvious from the VIL definition and construction of* β.

Proposition 4 states that if the j-th sequence of the SDB contains two sequences of size i, say α and γ, which differ only in the last itemset, then $VIL_\gamma[j] < VIL_\alpha[j]$ is a sufficient condition to prove that the j-th sequence also contains the extended sequence β of size $i+1$, obtained by juxtaposing a_i to γ.

Proposition 5 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$, $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$, $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, $VIL_\gamma[j] \neq null$, $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, a_{i+1}, \ldots, a_m \rangle$. *If* $VIL_{\alpha\epsilon}[j] \neq null$ *and* $VIL_\gamma[j] < VIL_\alpha[j]$, *then* $VIL_\beta[j] \neq null$.

Proof 5 *From proposition 2 and* $VIL_{\alpha\epsilon}[j] \neq null$ *it follows that* $VIL_\alpha[j] \neq null$. *Since conditions for proposition 4 hold, it follows that* $VIL_{\langle a_1, \ldots, a_{i-1}, b, a_i \rangle}[j] = VIL_\alpha[j]$. *From proposition 3 it follows that* $VIL_\beta[j] \neq null$.

Proposition 6 *Let* $\alpha = \langle a_1, \ldots, a_i \rangle$, $\epsilon = \langle a_{i+1}, \ldots, a_m \rangle$, $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$, $a_i \subset b$, $\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_m \rangle$. *If* $VIL_{\alpha\epsilon}[j] \neq null$ *and* $VIL_\gamma[j] = VIL_\alpha[j]$, *then* $VIL_\beta[j] \neq null$.

Proof 6 *It follows straightforwardly from proposition 3.*

For the sake of computational efficiency, our implementation of the VIL does not maintain the transaction-ids, but the pointers to the related SILs. In particular, $VIL_\alpha[j]$ points to $SIL_i[j]$ if the transaction-id i belongs to VIL_α.

Before building the CSET, the SILs of all frequent itemsets are incrementally computed. SILs for itemsets of size 1 are built during the first database scan. Then SILs of itemsets of size greater than 1 are built during the itemset-extension step (see Section 5.5.1).

In contrast, VILs are associated to frequent sequences and are built during the sequence-extension step (see Section 5.5.2). Initially, for each sequence α of size 1, VIL_α is straightforwardly computed from SIL_t, where t is the only closed itemset in α. In particular, for each $j \in \{1 \ldots n\}$, $VIL_\alpha[j]$ is the first value of the list $SIL_t[j]$.

Example 9 *Fig. 5.2(c) shows the VIL_α of the 1-itemset sequence $\alpha = \langle\{a\}\rangle$. The value $VIL_\alpha[j]$ corresponds to the* transaction-id *of the first occurrence of the itemset $\{a\}$ in the sequence S_j, which is actually stored in $SIL_{\{a\}}[1]$ (see Fig. 5.2(a)). Therefore, $VIL_\alpha = [1, 2, 2]$.*

The computation of the VILs for sequences of size greater than 1 is explained in Section 5.5.2.

5.5.1 I-Step: using SILs

The itemset extension step (*I-Step*) is executed during the construction of the CIET. Suppose we have two sparse id-lists SIL_{i_1} (for the itemset i_1) and SIL_{i_2} (for the itemset i_2) and we want to extend the itemset i_1 with items in i_2. The SIL of $i_1 \cup i_2$ ($SIL_{i_1 \cup i_2}$) can be obtained by simultaneously scanning all the rows of SIL_{i_1} and SIL_{i_2}. In particular, for each row j, only the transaction-ids which are found in both $SIL_{i_1}[j]$ and $SIL_{i_2}[j]$ are inserted in $SIL_{i_1 \cup i_2}[j]$. Thus, $SIL_{i_1 \cup i_2}$ represents the occurrences of the itemset $i_1 \cup i_2$ in the database.

For each itemset i, its support can be efficiently computed by counting the non-null vector elements in the SIL_i. This can be done during the construction of the SIL_i at no additional cost.

Example 10 *Consider the running example in Figure 5.1(c). Figures 5.2(a) and 5.1(a) show the sparse id-lists for itemsets $\{a\}$ and $\{b\}$. Figure 5.1(b) displays the sparse id-list for the itemset $\{a, b\}$. It contains for the first list (in row 1) only the element with value 1, for the second list the value null, and for the list in row 3 only the element with value 2. The support of itemset $\{a, b\}$ is 2, that is, the number of rows whose values are different from* null.

5.5.2 S-Step: using VILs

Consider two sibling nodes in the CSET and their corresponding sequences $\alpha = \langle a_1, a_2, \ldots, a_m\rangle$ and $\beta = \langle b_1, b_2, \ldots, b_m\rangle$. By constructing the CSET, we have that $a_m \neq b_m$ and $\forall i = 1, \ldots, m-1 : \ a_i = b_i$. The sequence extension step (*S-step*) of α using β aims at both constructing a new sequence $\gamma = \langle a_1, a_2, \ldots, a_m, b_m\rangle$ by appending b_m to α, and computing VIL_γ from VIL_α and VIL_β.

The computation of VIL_γ proceeds as follows. If either $VIL_\alpha[j]$ or $VIL_\beta[j]$ are *null*, i.e. α and β do not occur together in the j-th sequence in SDB, then $VIL_\gamma[j]$ is set to *null*, since γ cannot occur in the sequence itself. If both $VIL_\alpha[j]$ and $VIL_\beta[j]$ are non-null, we have to check that an occurrence of a_m

that precedes an occurrence of b_m in the j-th sequence exists. Procedurally, this is performed as follows. While $VIL_\beta[j] \neq null$ &&[1] $VIL_\alpha[j] \geq VIL_\beta[j]$, the reference to $SIL_{\{b_m\}}[j]$ stored in $VIL_\beta[j]$ is used to *right-shift* to the next transaction-id in $SIL_{\{b_m\}}[j]$. At the end, if $VIL_\alpha[j] < VIL_\beta[j]$, the transaction-id found (possibly after some right-shifts) in $SIL_{\{b_m\}}[j]$ is stored in $VIL_\gamma[j]$ (check succeeded), otherwise $VIL_\gamma[j]$ is set to *null* (check failed).

During the S-Step, only closed itemsets are considered in the sequences. This guarantees a significant reduction of the search space.

Example 11 *Consider the database in Figure 5.1(c). Let Figures 5.2(c) and 5.2(d) be the VILs for the sequences* $\alpha = \langle\{a\}\rangle$ *and* $\beta = \langle\{e\}\rangle$*, respectively. Figure 5.2(e) shows the VIL of sequence* $\gamma = \langle\{a\},\{e\}\rangle$ *resulting from an* S-step *on* α *using* β*.*

- *The initial values of* $VIL_\alpha[1]$ *and* $VIL_\beta[1]$ *are 1 and 3, respectively. Since* $VIL_\alpha[1] < VIL_\beta[1]$*,* $VIL_\gamma[1] = 3$*.*
- *The initial values of* $VIL_\alpha[2]$ *and* $VIL_\beta[2]$ *are 2 and 1, respectively. Since* $VIL_\alpha[2] \geq VIL_\beta[2]$*, the reference to* $SIL_{\{e\}}[2]$ *stored in* $VIL_\beta[2]$ *is used to identify the next transaction-id of* $\{e\}$ *(Figure 5.2(b)). Since this value does not exist,* $VIL_\gamma[2] = null$*.*
- *The initial values of* $VIL_\alpha[3]$ *and* $VIL_\beta[3]$ *are 2 and 1, respectively. Since* $VIL_\alpha[3] \geq VIL_\beta[3]$*, the reference to* $SIL_{\{e\}}[3]$ *stored in* $VIL_\beta[3]$ *is used to identify the next transaction-id of* $\{e\}$*, i.e. 3. This means that* $VIL_\gamma[3] = 3$*.*

In the next section, the importance of both the I-step and the S-step for the CloFAST algorithm is explained.

5.6 CloFAST: The algorithm

In this section I describe the CloFAST algorithm (see Algorithm 7) and the one-step technique used to simultaneously check for both sequence closure and sequence pruning.

With the first database scan, CloFAST finds the frequent 1-itemsets and builds their sparse id-lists (line 2). Then it simultaneously discovers the closed frequent itemsets and builds their sparse id-lists (line 4). This is achieved by building a CIET, based on a modified version of the algorithm FAST [SFMH11], which integrates the marking and pruning technique proposed in Moment [CWYM06].

[1] Here && denotes the shot-cut AND predicate.

Algorithm 7 CloFAST(SDB, min_sup)

Input: Sequence database SDB, int min_sup

Output: Complete set of closed freq. sequences CFS;

Data: CSET T=new Tree(), Frequent Items FI, Closed Frequent Itemset CFI, Node n;

1: *// Identify frequent 1-itemsets and build their SILs;*
2: FI = *loadFrequentSILs*(SDB, min_sup);
3: *// Identify closed frequent itemsets and their SILs*
4: CFI= *mineClosedFItemset*(FI, min_sup);
5: **for each** cfi ∈ CFI **do**
6: *// Create VIL_{cfi} from SIL_{cfi}*
7: vil=*createVil*(cfi);
8: *// Create CSET node associated to cfi*
9: n= *createNode*(cfi,vil);
10: *labelNodeAs*(n,"closed");
11: *addChildNode*(T,root(T),n);
12: **end for**
13: **for each** child ∈ *children*(T,root(T)) **do**
14: *// s*tart the depth first search
15: *sequenceExtension*(T,child,min_sup);
16: **end for**
17: **return** *closedSequentialPatterns*(T);

The first level of the CSET is initialized in lines 5-12. Each node in the first level represents a (candidate) closed sequence of size 1, whose unique element is a closed frequent itemset. The VILs of the nodes at the first level are straightforwardly computed from the SILs of the closed frequent itemsets. Starting from the first level, the nodes in the CSET are considered for sequence extension (lines 13-16) according to a depth-first search strategy.

During the mining process the current set of closed sequential patterns is stored in the CSET. At the end, CloFAST returns the complete set of closed sequential patterns in the CSET.

Algorithm 8 describes the sequence extension step for an input node n. It first tests if n is closed and/or can be pruned (line 1). This is achieved by means of the `checkClosureAndPrune` method detailed in the next subsections. If n is not pruned (line 2), for each of its siblings including itself, the *S-step* is executed, in order to generate its children sequences (lines 6-20). Only the new frequent sequences are stored in the CSET, together with their corresponding

Algorithm 8 SequenceExtension(T, n, min_sup)

Input: CSET T, Node n, int min_sup;

Data: Node u, newNode, child; VIL v1,v2,v3; Sequence newSequence; int supp

```
1: checkClosureAndPrune(n, T);
2: if pruned(n); then
3:   return
4: end if
5: v1 = Vil(n);
6: for each u ∈ siblings(T,n) do
7:   v2 = Vil(u);
8:   (v3,supp) = S-Step(v1,v2); // create the new vertical id-list and compute
     its support
9:   if supp ≥ min_sup then
10:    if supp = support(v1); then
11:      labelNodeAs(n,"nonClosed");
12:    end if
13:    // create new CSET node
14:    newSequence =extend(sequence(n), sequence(u));
15:    newNode = createNode(newSequence, v3);
16:    labelNodeAs(newNode,"closed");
17:    addChildNode(T, n, newNode);
18:  end if
19: end for
20: for each child ∈ children(T, n); do
21:   sequenceExtension(T, child, min_sup);
22: end for
```

VILs (denoted as $v3$ in the algorithm). If a generated sequence has the same support as that represented in n, then n is labeled as *non-closed* (lines 11-13), otherwise it is labeled as *closed* by default (it is indeed a candidate closed frequent sequence). In lines 21-23 the sequence extension step is recursively applied to each child of n.

5.6.1 Backward Closure Checking

Inspired by BIDE [WHL07], I aim at pruning the search space by exploiting a closure checking schema which, besides the forward construction of the CSET, operates in a backward fashion. Closure checking is important since it is useless to further explore a node if this node, and its descendants, could be absorbed

by nodes present in other paths of the tree.

The intuition behind the backward solution is that it would lead to first check sequences in the tree which are more "similar" to the sequence α to be evaluated (same head, same length, closer in the tree). This means that, if a sequence which absorbs α exists, the backward closure checking is faster than a classical top-down solution. If there is no such sequence, the two approaches are equivalent.

Methods for pruning frequent closed itemsets have already been presented in the literature [CWYM06]. However, search-space pruning in closed frequent sequence mining is trickier than in closed frequent itemset mining. Indeed, while a depth-first-search-based closed itemset mining algorithm can *safely* stop growing a prefix itemset as soon as it finds that this itemset can be absorbed by another closed itemset already generated, a closed sequence mining algorithm needs additional checks. This is due to both the possible presence of multiple instances of the same itemset in a sequence and to the ordering among the itemsets in the sequence.

Pruning is rather complex in BIDE, since it is based on pseudo-projected databases. On the contrary, CloFAST takes advantage of the VIL data structures, which convey essential information for pruning. Indeed, it is useless to expand a node n if, in other branches of the tree a node exists that has the same VIL as n and represents a sequence which is a supersequence of that represented in n.

This is described in Algorithm 9. Given a CSET node n representing a sequence $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$, `checkClosureAndPrune` first checks whether α is closed. If not, it checks whether n can be safely pruned. As defined in Section 5.2, α is non-closed if any supersequence β of α exists that absorbs α. This definition can be used for *backward closure*.

Definition 18 Backward Closure.

Let $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ be a frequent sequence of size m. Then α is non-closed in backward closure if for some $i \in \{1 \ldots m\}$ an itemset b exists, such that one of the following two conditions holds:

1. *$a_i \subset b$ and $\langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_m \rangle$ has the same support as α (itemset closure);*

2. *$\langle a_1, \ldots, a_{i-1}, b, a_i, \ldots, a_m \rangle$ has the same support as α (sequence closure).*

Given the sequence α represented in the node n, Algorithm 9 identifies an itemset b which allows us to check whether α is non-closed in backward closure

Algorithm 9 checkClosureAndPrune(n, T)

Input: Node n, CSET T;

Data: Node u; List siblings ; VIL vilU, vilP;

```
1: i = level(n);
2: n'= parent(T,n);
3: repeat
4:    vilP =  Vil(n');
5:    children = children(T, n');
6:    for each u ∈ children do
7:       vilU =  Vil(u);
8:       // check if last itemset of u contains the i-th itemset of n
9:       if contains(lastItemset(u), itemset(n,i)) then
10:         if itemsetClosure(vilU, path(n', n)) then
11:            labelNodeAs(n,"nonClosed");
12:            if earlyTermination(vilU,vilP) then
13:               labelNodeAs(n,"pruned");
14:            end if
15:            return;
16:         end if
17:      end if
18:      if sequenceClosure(vilU, path(n', n)) then
19:         labelNodeAs(n,"nonClosed");
20:         if earlyTermination(vilU,vilP) then
21:            labelNodeAs(n,"pruned");
22:         end if
23:         return;
24:      end if
25:   end for
26:   i= i-1;
27:   n' = parent(T,n');
28: until n' ≠ root(T);
```

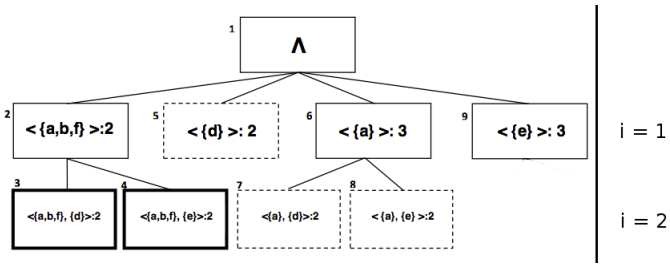


Figure 5.5: A partial view of the CSET for the dataset in Figure 5.1(c).

or not. The search can be safely restricted to the last itemset of the sequences represented in the siblings of either n or its ancestors. This is done by using only the CSET, which is climbed level-by-level, starting from the direct parent n' of n (line 2) and considering each child u of n' (line 6). The algorithm identifies candidate sequences in the form of $\gamma = \langle a_1, \ldots, a_{i-1}, b\rangle$, represented in u. Such candidate sequences are then used in order to evaluate the support of either $\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_m\rangle$, if $a_i \subset b$, or $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, \ldots, a_m\rangle$, in any case. It is noteworthy that, during the identification of candidate sequences in the form of γ, neither is the CSET modified nor are new sequences evaluated. Moreover, the computation of the support of the sequences β only exploits VILs, as I will explain later.

Examples 12 and 13 clarify these aspects for the two cases in Definition 18.

Example 12 ***(Itemset closure).***

Consider the dataset in Figure 5.1(c) and the sequence $\alpha = \langle\{a\}, \{d\}\rangle$ represented in node 7 of Figure 5.5. CloFAST examines at the first step (level $i = 2$) the sequence $\gamma = \langle\{a\}, \{e\}\rangle$ represented in node 8 (sibling of 7). The last itemset of γ (i.e. $\{e\}$) does not contain the last itemset of α (i.e. $\{d\}$), so the itemset closure cannot be checked. CloFAST moves at the previous level ($i = 1$) and checks the itemset closure of α over the itemset $\{a\}$ using the children of the CSET's root (nodes 2, 5, 6, 9). Given $\gamma = \langle\{a, b, f\}\rangle$ (node 2), since the last itemset of γ contains the last but one itemset of α (i.e. $\{a\}$), CloFAST checks whether the supersequence $\beta = \langle\{a, b, f\}, \{d\}\rangle$ can absorb α. In that case, α is labeled as non-closed.

Example 13 ***(Sequence closure).***

Consider the dataset in Figure 5.1(c) and the sequence $\alpha = \langle\{e\}, \{d\}\rangle$ (see node 12 in Figure 5.6). CloFAST examines at the first step (level i=2) the children of the parent of α, i.e. the sequence $\gamma = \langle\{e\}, \{a\}\rangle$ (node 10). By inserting $\{a\}$, the last itemset of sequence γ, between the itemsets $\{e\}$ and $\{d\}$

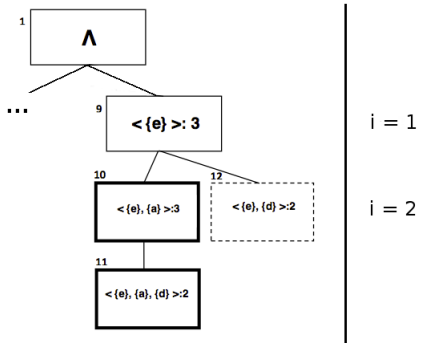


Figure 5.6: A partial view of the CSET for the dataset in Figure 5.1(c).

of α, we obtain the sequence $\beta = \langle\{e\},\{a\},\{d\}\rangle$ (node 11), which absorbs α. Thus, α is labeled as non-closed.

As previously mentioned, backward closure is verified by only working on the CSET and VILs. Algorithmically, the predicate `contains` (line 9) checks whether the i-th itemset of α is a proper subset of the last itemset in γ. In this case, the predicate `itemsetClosure` is executed to check whether β absorbs α. If the `itemsetClosure` is false, CloFAST checks the `sequenceClosure` predicate (line 18).

In order to explain how backward closure is verified in CloFAST, I define two predicates, namely $shiftSC$ (shift Sequence-Closure) and $shiftIC$ (shift Itemset-Closure). The former (latter) works on the VILs and SILs to check whether whenever the j-th sequence in SDB contains α it also contains the supersequence β constructed as in Definition 18 - case 2 (case 1). Obviously, the opposite is always true, i.e. whenever the j-th sequence in SDB contains the supersequence β it also contains the subsequence α. Therefore, if either $shiftSC$ or $shiftIC$ hold for each j, then α and β have the same support, i.e. β absorbs α (or α is non-closed).

Definition 19 ***(shiftSC)***. *Let $\alpha = \langle a_1,\ldots,a_{i-1},a_i,\ldots a_m\rangle$ be the frequent sequence for which we intend to verify `sequenceClosure` (at the i-th level), $\delta = \langle a_1,\ldots,a_{i-1},a_i\rangle$ be the $(m-i)$-th ancestor of α and $\gamma = \langle a_1,\ldots,a_{i-1},b\rangle$ be a sibling of δ. Let the j-th sequence in SDB contain α, i.e. $VIL_\alpha[j] \neq null$. Then, the predicate ShiftSC, which takes as input both $VIL_\gamma[j]$ and the list of the VILs stored in the path from δ to α,[2] is recursively defined as follows:*

[2]In order to simplify the notation, I will use the term sequence to identify the CSET node that contains the sequence itself.

$$shiftSC(VIL_\gamma[j], [VIL_\delta[j], VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], VIL_{\langle a_1,\ldots,a_{i+2}\rangle}[j], \ldots, VIL_\alpha[j]]) =$$

$$\begin{cases} true & if \quad \left(\begin{array}{l} VIL_\gamma[j] < VIL_\delta[j] \\ \vee \quad \exists t_{a_i} \in SIL_{a_i}[j], t_{a_i} \neq null \text{ such that} \\ \qquad \big(VIL_\gamma[j] < t_{a_i} \wedge shiftSC(t_{a_i}, [VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], \ldots, VIL_\alpha[j]])\big) \end{array} \right) \\ false & otherwise \end{cases}$$

Thus, $shiftSC$ checks whether $VIL_\gamma[j] < VIL_\delta[j]$, i.e. b, the last itemset of γ, can precede a_i, the last itemset of δ in the j-th sequence. If so, from proposition 5 the j-th sequence in SDB contains the supersequence $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, \ldots a_m\rangle$. If not, the check is repeated on a virtual shift to the next transaction-id in the list $SIL_{a_i}[j]$. This amounts to determining an alternative value, if any, for $VIL_\delta[j]$, such that $VIL_\beta[j] \neq null$, i.e. the sequence $\langle a_{i+1}, \ldots a_m\rangle$ can be juxtaposed to $\langle a_1, \ldots, a_{i-1}, b, a_i\rangle$ by preserving the containment relationship for the j-th sequence.

If the conditions stated in Definition 19 are satisfied for all non-null values of VIL_α, then α is labeled as *non-closed*.

Definition 20 ***(shiftIC)***. *Let $\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m\rangle$ be the frequent sequence for which we intend to verify the* `itemsetClosure` *(at the i-th level), $\delta = \langle a_1, \ldots, a_{i-1}, a_i\rangle$ be the $(m-i)$-th ancestor of α and $\gamma = \langle a_1, \ldots, a_{i-1}, b\rangle$ be a sibling of δ, such that $a_i \subset b$. Then, the predicate $ShiftIC$, which takes as input $VIL_\gamma[j]$ and the list of the VILs stored in the path from δ to α, is defined as follows:*

$$shiftIC(VIL_\gamma[j], [VIL_\delta[j], VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], VIL_{\langle a_1,\ldots,a_{i+2}\rangle}[j], \ldots, VIL_\alpha[j]]) =$$

$$\begin{cases} true & if \quad \left(\begin{array}{l} VIL_\gamma[j] = VIL_\delta[j] \\ \vee \quad \exists t_{a_i} \in SIL_{a_i}[j], t_{a_i} \neq null \text{ such that} \\ \qquad \big(t_{a_i} = VIL_\gamma[j] \wedge shiftSC(t_{a_i}, [VIL_{\langle a_1,\ldots,a_{i+1}\rangle}[j], \ldots, VIL_\alpha[j]])\big) \end{array} \right) \\ false & otherwise \end{cases}$$

Thus, *shiftIC* checks whether $VIL_\gamma[j] = VIL_\delta[j]$, i.e. the last itemset of δ can be replaced by the last itemset of γ. If so, from proposition 6 the j-th sequence in SDB contains the supersequence $\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots a_m\rangle$. If not, the check is repeated on a virtual shift to the next transaction-id in the

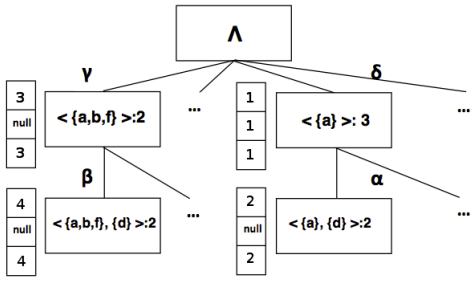


FIGURE 5.7: An example of itemset closure for sequence $\alpha = \langle\{a\}, \{d\}\rangle$. On the left of each node the corresponding VIL is shown.

list $SIL_{a_i}[j]$, which amounts to determining an alternative value, if any, for $VIL_\delta[j]$ such that $VIL_\beta[j] \neq null$.

It is noteworthy that the definition of $shiftIC$ depends on $shiftSC$, since we need to check that the rest of the sequence $\langle a_{i+1}, \ldots a_m\rangle$ can be juxtaposed to $\langle a_1, \ldots, a_{i-1}, b\rangle$ by preserving the containment relationship for the j-th sequence. If the conditions stated in Definition 20 are satisfied for all non-null values of VIL_α, then α is labeled as *non-closed*.

Since *shiftSC* and *shiftIC* coincide, apart from the test on the VILs ($<$ for shiftSC and $=$ for shiftIC), I show an example only for the *shiftIC* predicate.

Example 14 *Consider the following database SDB:*

1. $\langle\{a\}, \{d\}, \{a, b, f\}, \{d\}\rangle$,
2. $\langle\{a\}, \{c\}\rangle$,
3. $\langle\{a\}, \{d\}, \{a, b, f\}, \{d\}\rangle$,

and the sequence $\alpha = \langle\{a\}, \{d\}\rangle$. A partial view of the corresponding CSET is reported in Figure 5.7. According to the definition of `itemsetClosure`*, α is non-closed if, for each $j \in [1, \ldots, n]$, such that $VIL_\alpha[j] \neq null$, the predicate shiftIC is true. Since at level 2 (last level) there is no child whose last itemset can replace the last itemset of α, CloFAST moves up to the previous level and analyzes the children of the root. At this level, CloFAST checks whether the first itemset of α, i.e. $\{a\}$, can be replaced by the last itemset of the sequence γ, i.e. $\{a, b, f\}$. Consider the first sequence, i.e. $j = 1$. Since $VIL_\delta[1] = 1$ differs from $VIL_\gamma[1] = 3$, CloFAST checks whether it is possible to shift to the next transaction-id in the list $SIL_{\{a\}}[1]$. This leads to a virtual "shifting" of transaction-ids of $VIL_\delta[j]$ until $VIL_\delta[1] = VIL_\gamma[1]$ is satisfied. This is true since $SIL_{\{a\}}[1] = [1, 3]$. As a second step, CloFAST checks that the "new" value*

of $VIL_\delta[1]$, i.e. 3, is less than $VIL_\alpha[1]$, i.e. 2. Since this check fails, CloFAST performs a virtual "shifting" of $VIL_\alpha[1]$ using $SIL_d[1] = [2, 4]$ and obtains a "new" value of $VIL_\alpha[1]$, namely 4, such that $VIL_\delta[1] < VIL_\alpha[1]$ $(3 < 4)$. Since for each j such that $VIL_\alpha[j] \neq null$, i.e. j=1,3, the predicate shiftIC holds, α is labeled as non-closed.

The theoretical motivation for itemset closure checking originates from the following theorem.

Theorem 1 *itemsetClosure*

Let

- *SDB be a sequence database,*
- *$\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m \rangle$ be the frequent sequence in SDB to be checked for itemset closure on the i-th itemset a_i,*
- *$\delta = \langle a_1, \ldots, a_{i-1}, a_i \rangle$ be the $(m - i)$-th ancestor of α in the CSET,*
- *$\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$ be a sibling of δ such that $a_i \subset b$, and*
- *$\beta = \langle a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots a_m \rangle$ be a supersequence of α.*

If:

$$\forall j = 1, \ldots, n : (VIL_\alpha[j] \neq null \Rightarrow shiftIC(VIL_\gamma[j], [VIL_\delta[j], \ldots, VIL_\alpha[j]])) \quad (5.1)$$

then β absorbs α.

Proof 7 *By definition of $shiftIC$, if $VIL_\alpha[j] \neq null$ and $shiftIC(VIL_\gamma[j], [VIL_\delta[j], \ldots, VIL_\alpha[j]])) = true$, then $VIL_\beta[j] \neq null$. Thus, sequences that contain α also contain β. Vice versa, since β is a supersequence of α, sequences that contain β also contain α. This means that the support of α is the same as the support of β, i.e. β absorbs α.*

Theorem 2 provides sufficient conditions for sequence closure checking.

Theorem 2 *sequenceClosure*

Let

- *SDB be a sequence database,*
- *$\alpha = \langle a_1, \ldots, a_{i-1}, a_i, \ldots a_m \rangle$ be the frequent sequence in SDB to be checked for sequence closure on the i-th itemset a_i,*

- $\delta = \langle a_1, \ldots, a_i \rangle$ *be the* $(m-i)$*-th ancestor of* α*,*
- $\gamma = \langle a_1, \ldots, a_{i-1}, b \rangle$ *be a sibling of* δ *and*
- $\beta = \langle a_1, \ldots, a_{i-1}, b, a_i, \ldots a_m \rangle$ *be a supersequence of* α*.*

If:

$$\forall j = 1, \ldots, n : (VIL_\alpha[j] \neq null \Rightarrow shiftSC(VIL_\gamma[j], [VIL_\delta[j], \ldots, VIL_\alpha[j]])) \quad (5.2)$$

then β *absorbs* α*.*

Proof 8 *The proof is analogous to that of Theorem 1.*

5.6.2 Pruning

If a node n is labeled as *non-closed*, then it is evaluated for pruning (Algorithm 9, lines 12-13 and 21-22). Indeed, it is possible that a *non-closed* sequence can still be profitably used for generating closed sequences. As described in Example 14, the sequence $\alpha = \langle \{a\}, \{d\} \rangle$ is *non-closed* because $\beta = \langle \{a, b, f\}, \{d\} \rangle$ absorbs α. However, it can be used to generate, in sequence extension, the closed sequence $\langle \{a\}, \{d\}, \{a, b, f\} \rangle$, which cannot be generated if the subtree rooted in the node associated to the sequence α is pruned.

On the other hand, there are cases in which *non-closed* sequences cannot lead to the generation of closed sequences. In these cases, their corresponding nodes should be labelled as pruned, in order to prevent CloFAST from generating further unpromising patterns. The following proposition provides the theoretical basis for pruning.

Proposition 7 *Let* $\alpha = \langle a_1, a_2, \ldots, a_m \rangle$ *be a frequent sequence,* $\beta = \langle b_1, b_2, \ldots, b_p \rangle$ *a supersequence of* α *and* N *the number of the elements in the* VIL_α *which differ from the corresponding transaction-ids in the* VIL_β*. If* $N = 0$*, i.e.* $VIL_\alpha = VIL_\beta$*, then for the two sequence extensions* $\gamma = \langle a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_q \rangle$ *and* $\delta = \langle b_1, b_2, \ldots, b_p, c_1, c_2, \ldots, c_q \rangle$*,* $VIL_\gamma = VIL_\delta$*.*

Proof 9 *By induction on* q*.*

- Base case*:* $q = 0$*. Trivial.*
- Induction step*:* $q > 0$*. Consider the two sequences* $\alpha' = \langle a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_{q-1} \rangle$ *and* $\beta' = \langle b_1, b_2, \ldots, b_p, c_1, c_2, \ldots, c_{q-1} \rangle$*. By construction,* β' *is a supersequence of* α'*. If* $N = 0$*, by inductive hypothesis,* $VIL_{\alpha'} = VIL_{\beta'}$*. If we juxtapose the same itemset* c_q *to both sequences, the VILs of the two extended sequences* $\gamma = \langle a_1, a_2, \ldots, a_m, c_1, c_2, \ldots, c_q \rangle$ *and* $\delta = \langle b_1, b_2, \ldots, b_p, c_1, c_2, \ldots, c_q \rangle$*, will still be the same, i.e.* VIL_γ=VIL_δ*.*

It is noteworthy that γ and δ have the same support. Since δ is a supersequence of γ, then δ absorbs γ. Therefore, it is useless to generate the extensions of α, since they will be absorbed by the sequences generated from β (*early termination condition*).

In CloFAST this early termination condition is efficiently checked during both the *itemset closure* and *sequence closure* phases (lines 12, 20) at no additional cost. In particular, if for each j such that $VIL_\alpha[j] \neq null$, the predicates *shiftIC* or *shiftSC* are satisfied without applying the virtual "shifting", then the node representing α can be labeled as pruned.

Example 15 *Consider the itemset closure described in Example 12 and depicted in Figure 5.5. At the first level (i=1), CloFAST applies the ShiftIC predicate having as arguments the VIL of node 2 ($\gamma = \langle\{a, b, f\}\rangle$) and the VILs of the path between nodes 6 ($\delta = \langle\{a\}\rangle$) and 8 ($\alpha = \langle\{a\}, \{d\}\rangle$). Since, for each j such that $VIL_\alpha[j] \neq null$ (i.e. $j = 1$ and $j = 3$), $VIL_\gamma[j] = VIL_\delta[j]$ (in particular, $VIL_\gamma = VIL_\delta = [1, 2, 2]$), then the sequence $\beta = \langle\{a, b, f\}, d\rangle$ exists that has the same VIL as α and will generate the same supersequences as α. For this reason node 2, which represents the sequence α, can be labeled as pruned.*

Recalling that the backward closure is verified by only working on VILs, the time complexity of Algorithm 9 is $O(d \cdot |SDB|)$, where d is the depth of the tree. Therefore, the backward closure can be efficiently checked in practical cases characterized by relatively small values of d.

5.7 Experiments

In order to empirically evaluate CloFAST, in this section I report the experimental results on both real world and synthetic datasets. I implemented CloFAST in Java and compared it with BIDE, ClaSP and CloSpan provided by the Java framework SPMF [FV][3]. The experimental setting is inspired by [WHL07] and aims at evaluating:

- **Efficiency:** Efficiency is evaluated both in terms of running time (seconds) and memory consumption (Gb) on sparse and dense datasets. Following [GCMG13], I define the density as the ratio between the average number of items in an itemset and the number of different items. When this value is small, the generated dataset is considered sparse, whereas,

[3]Unfortunately, I were not able to compare CloFAST with COBRA, which is not publicly available. Moreover, the algorithm description provided in [HCTH06] does not provide enough details to unambiguously re-implement the system.

when this ratio is high the dataset is considered dense. I compare CloFAST efficiency both on synthetic and real datasets.

- **Scalability:** CloFAST is compared with the above cited algorithms by linearly increasing the number of input sequences. I report the results in terms of running time (seconds) and memory consumption (MB). Scalability is only evaluated on artificially generated datasets.

- **Effectiveness of the CloFAST optimization technique:** CloFAST is compared with FAST which does not implement the *backward closure checking* and *pruning* techniques. I report results in term of running time (seconds), memory consumption (GB) and number of mined frequent patterns. This comparison is performed on real datasets.

All the results reported in this section are obtained with a machine with a 4-core 2.4GHZ Intel Xeon processor, running Ubuntu 12.04 Server edition with 32GB of main memory. In order to facilitate the replication of the experiments, the system and all the considered datasets can be downloaded at the following hyperlink: `http://www.di.uniba.it/~ceci/micFiles/systems/CloFAST/`

Before presenting the results obtained, I describe the datasets used in the experiments.

5.7.1 Dataset description

The synthetic datasets used for our experiments were obtained using the IBM data generator [AS95]. This dataset generator has been used in most sequential pattern mining studies [Zak01, PHMA$^+$01, AS95, GCMG13]. Generated datasets contain random sequences of itemsets which can be easily controlled by the user. In particular, the generator allows the user to specify several parameters which regulate, among other aspects, the number of sequences, the average number of transactions per sequence and the number of different items. The detailed list of parameters used in this evaluation is listed and explained in Table 5.2. The parameter values are reported in the following subsections and depend on the specific purpose of each empirical evaluation.

I also compared the algorithms on real datasets, that is, Gazelle, Snake, MSNBC and Pumsb. For all the datasets, except Snake, I also considered variants which are commonly used in the literature. I indicate such variants with the star (*) suffix. The properties of all the real datasets used in our experiments are reported in Table 5.3 and described in the following:

Parameter	Description
D	Number of sequences ($*10^3$)
C	Average number of itemsets per sequence
T	Average number of items per itemset
S	Average length of maximal sequences
I	Average size of itemsets in maximal sequences
N	Number of different items ($*10^3$)

Table 5.2: Parameters used in the IBM data generator. In the definition of S and I, a sequence is considered maximal if it is not a sub-sequence of any other frequent sequence [Zak01].

- Gazelle (BMS-WebView-1) is a dataset used in the KDDCup-2000 competition and, basically, it includes a set of page views done by users on the `gazzelle.com` e-commerce web site. Product pages viewed in one session are considered an itemset, and different sessions for one user define the sequence. Gazelle* represents another version of the dataset proposed in the KDDCup-2000 competition and used in past studies on sequential pattern mining [WHL07]. Both datasets are considered sparse datasets. Gazzelle was downloaded from `www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php`, while Gazzelle* was downloaded from the KDD Cup 2000 web site.

- MSNBC is a dataset of click-stream data (from the UCI repository). They are collected from logs of `msnbc.com` and news-related portions of `msn.com` for the entire day of September 28th, 1999. Each sequence in the dataset corresponds to page views of a user during that twenty-four hour period. Each transaction in the sequence corresponds to a user's request for a page. MSNBC was downloaded from `http://archive.ics.uci.edu/ml/datasets/MSNBC.com+Anonymous+Web+Data`, while MSNBC* has been downloaded from `www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php`.

- Snake is a biological dataset which contains 192 Toxin-Snake protein sequences and 20 unique items. This Toxin-Snake dataset is about a family of eukaryotic and viral DNA binding proteins and was used in [WHL07]. For our experiments only sequences containing more than 50 items were kept. This filtering is performed in order to make the dataset more uniform (because the original Snake dataset contains only a few very short se-

Dataset	# seq.	avg length	max length	# items	density
Gazelle	59,601	2.51	267	497	0.002
MSNBC	989,818	4.70	14,795	17	0.06
Pumsb	49,046	50.48	63	2,088	0.0005
Gazelle*	29,369	2.98	651	1,423	0.0007
Snake*	163	6.62	61	21	0.04
MSNBC*	31,790	13.33	100	17	0.06
Pumsb*	9,230	50.49	61	1,676	0.0006

Table 5.3: Properties of the real datasets considered for the experiments.

quences and many long sequences). The dataset obtained (called Snake*) contains 163 long sequences with an average of 60.62 items. This dataset is not publicly available.

- Pumsb contains census data for population and housing from PUMS (Public Use Microdata Sample) [BCF+05]. Both Pumsb and Pumsb* was downloaded from `http://fimi.ua.ac.be/data/`.

5.7.2 Results: Efficiency of CloFAST on synthetic datasets

As previously stated, to test the efficiency of CloFAST I adopted the schema based on sparse and dense datasets proposed by Gomariz *et al.* [GCMG13]. They showed how the performance of the sequential pattern mining algorithms largely depends on the database density, and they introduced a definition of density based on T/N (see Tab. 5.2). When T/N is small, the generated dataset is sparse, while when T/N grows, the dataset tends to be dense.

To evaluate and compare the efficiency of the algorithms, I considered four configurations. In the first, I fixed $D = 5$ (number of transactions $*10^3$), C=10 (the sequence length), T=10 (number of items in an itemset) and varied N (the number of different items). I obtained the datasets D5C10T10N2.5S6I4, D5C10T10N1.6S6I4 and D5C10T10N1S6I4. In the second, I fixed D=50, C=20, N=2.5 and varied T, obtaining the datasets D50C20T10N2.5S6I4, D50C20T20N2.5S6I4, D50C20T30N2.5S6I4 and D50C20T40N2.5S6I4, which are denser than the datasets belonging to the first configuration.

In Figure 5.8 I compare CloFAST with ClaSP, BIDE and CloSpan in terms of the running time (in seconds) and memory consumption (in GB), according to the first dataset configuration and varying the support threshold. In terms of running time (graphics are reported in logarithmic scale), CloFAST generally outperforms all the other systems, especially for low support values, when the number of frequent sequences is higher. By increasing the density of the dataset (i.e. by decreasing N) the advantage of CloFAST over the other three

algorithms becomes more evident. Since the higher density is directly related to the number of frequent sequences, we can conclude that the higher the number of frequent sequences, the more competitive (in running time) the proposed algorithm. Notably, the time efficiency of CloFAST is not obtained at the cost of higher memory consumption, which remains comparable to that of CloSpan. For highly dense datasets and for small values of the support threshold, the worst performing system is BIDE. This is probably related to the fact that, for dense datasets, the size of the projected databases does not shrink during the mining process. The situation is more favorable to BIDE for very sparse datasets and for small values of the support threshold, thus confirming the conclusions reported in [WHL07].

In Figure 5.9 I show the results obtained according to the second dataset configuration (i.e. by varying T) and setting the support threshold to 0.4. They confirm the discussion reported for Figure 5.8, particularly that CloFAST outperforms the algorithms BIDE, ClaSP and CloSpan when the density of the datasets increases. It is noteworthy that ClaSP does not return results with the dataset D50C20T40N2.5S6I4 ($T/N = 16$), since it consumes all the assigned memory (fixed to 32GB).

Moreover, the efficiency of CloFAST with distinct density values is evaluated by varying the number of itemsets in the sequences (C). In Figure 5.10 I show the running time and memory consumption of the considered algorithms using a third and a fourth dataset configuration. For the sparsest configuration (T=2.5, N=10, D=20), I compare the performances obtained with four datasets (D20C20T2.5N10S6I4, D20C40T2.5N10S6I4, D20C60T2.5N10S6I4, D20C80T2.5N10S6I4) and 2 support thresholds. For the densest configuration (T=20, N=4, D=10), I obtained the datasets D10C20T20N5S6I4, D10C40T20N5S6I4, D10C60T20N5S6I4, D10C80T20N5S6I4 and showed the results only for one support threshold. We observe that for the densest configuration it was not possible to test lower support thresholds, due to the extremely large number of frequent sequences. The results show that, in general, by increasing the number of itemsets in the sequences (C), CloFAST shows lower running times than other systems. This behavior is more evident for the more complex task of mining dense datasets with a high number of itemsets in the sequences (and a high number of frequent patterns). In this case, CloFAST outperforms competitors by one order of magnitude (see Figure 5.10 (c)), while keeping memory consumption under control (see Figure 5.10 (f)). Concerning this last aspect, we observe again a good behavior of CloSpan in terms of memory consumption. This effect is explained by the efficient way CloSpan stores internal data structures (integer vectors),

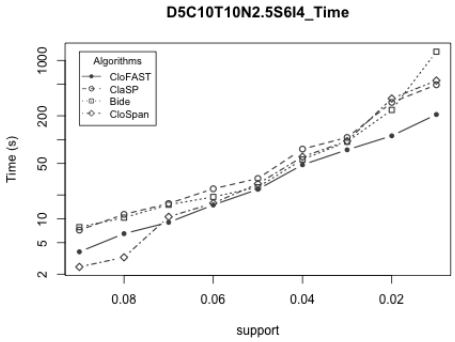
which allows it to save memory at the price of higher running times (note that running times are expressed in logarithmic scale, while memory consumption is expressed in linear scale).

Finally, I selected one experiment from the first, the second and the fourth configuration (median of values of other parameters) and varied S and I, obtaining the datasets D5C10T20N1.6S[2..10]I[2..10], D50C20T20N2.5S[2..10]I[2..10] and D10C60T20N5S[2..10]I[2..10]. In this way, it was possible to evaluate how the parameters S and I affected the computation time on the selected datasets. In Figures 5.11 and 5.15, I report the results obtained. From the twelve heatmaps, we can conclude that CloFast has the same trend as other algorithms but, coherently with the results reported before, it is the best performing in the case of dense datasets. In particular, on dense datasets, CloFast outperforms competitors by a good margin when the values of I and S are small (top-left corner of the heatmap), i.e., when the number of frequent patterns is higher.

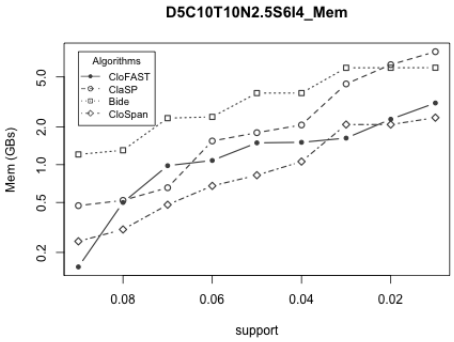
5.7.3 Results: Efficiency of CloFAST on real datasets

The results obtained on real datasets generally confirm the observations drawn from the experiments performed on synthetic datasets. In particular, the running times shown in Figure 5.12 confirm that CloFAST outperforms all the other methods when the support threshold is low, i.e. the number of frequent patterns is high. In particular, for MSNBC, MSNBC* and Snake*, which are the densest datasets (see Table 5.3), CloFAST clearly shows the best performance in running time. We note that for the datasets Pumbs and Pumbs*, it is difficult to appreciate the difference between CloFAST, ClaSP and CloSpan, since the high running time of BIDE flattes the other results. Nevertheless, we confirm that for these two datasets, CloFAST is the fastest algorithm.

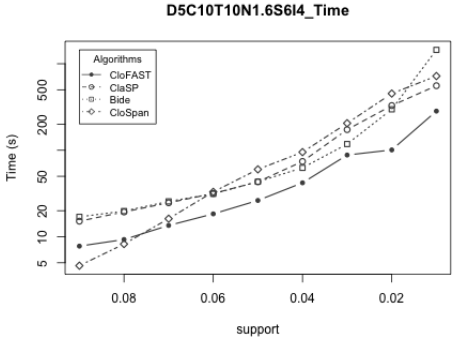
Concerning memory consumption, CloFAST is among the best performing methods for almost all the datasets. Some differences between CloFAST and ClaSP can be appreciated for two datasets with a large number of closed patterns ($294,386$ for MSNBC* with $min\_sup = 0.005$, $1,300,529$ for Snake* with $min\_sup = 0.5$). Both algorithms use a vertical representation of the data. However, a closer look at the results reveals that, while for MSNBC* CloFAST is at a disadvantage compared to ClaSP, due to the large number of null values stored in the VILs, for Snake* our internal representation is effective and CloFAST is the only method which does not incur in out-of-memory errors (the limit is 32GB).



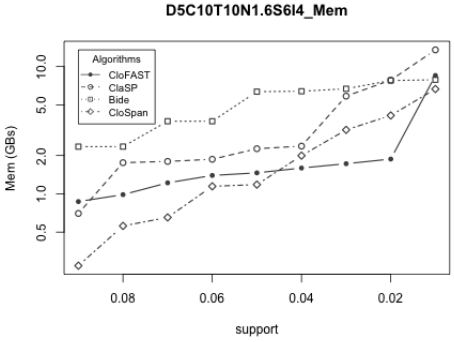
(a) D5C10T10N2.5S6I4



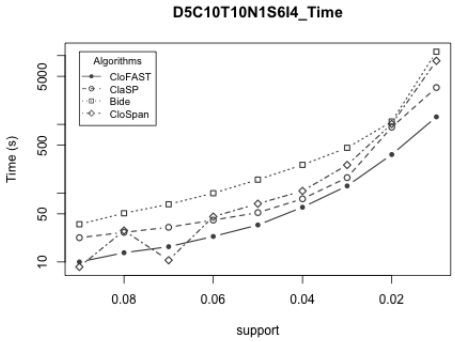
(d) D5C10T10N2.5S6I4



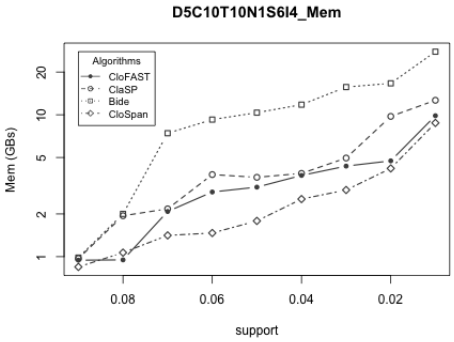
(b) D5C10T10N1.6S6I4



(e) D5C10T10N1.6S6I4



(c) D5C10T10N1S6I4



(f) D5C10T10N1S6I4

Figure 5.8: Running times (in seconds) and memory consumption (in Gb) varying $N = \{2.5, 1.6, 1\}$ and *min_sup*. Results are obtained with $D = 5$, $C = 10$ and $T = 10$.

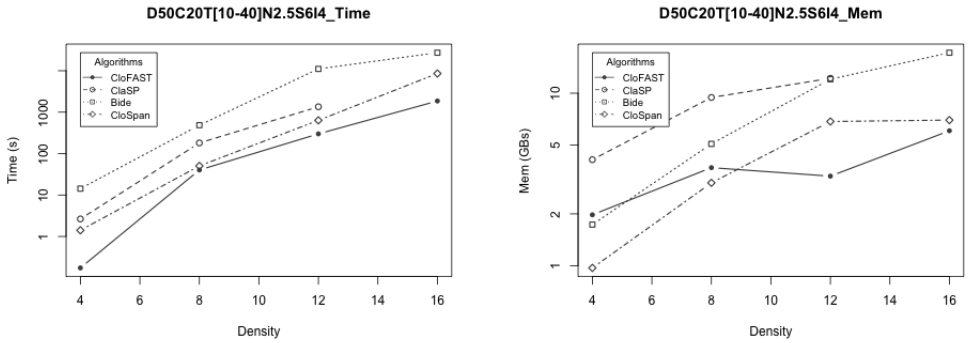


Figure 5.9: Running times (in seconds) and memory consumption (in Gb) varying $T/N = \{4, 8, 12, 16\}$. Results are obtained with $min\_sup = 0.4$, D=50, C=20, N=2.5.

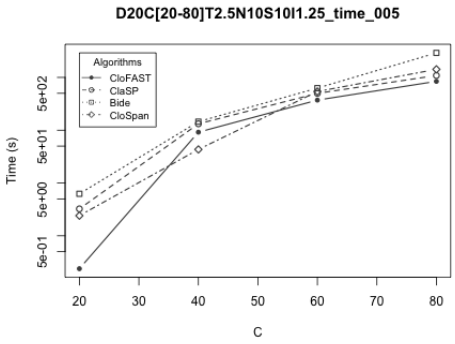
5.7.4 Scalability

In order to evaluate the scalability of CloFAST with respect to other competitive systems I performed experiments on synthetic datasets by varying the number of input sequences. In particular, by keeping constant other parameters of the data generator (C=20, T=20 and N=2.5), I varied D, obtaining datasets with a different number of sequences (i.e. the following configurations were used: D50C20T20N2.5S6I4, D100C20T20N2.5S6I4, D150C20T20N2.5S6I4, D200C20T20N2.5S6I4, D250C20T20N2.5S6I4, D300C20T20N2.5S6I4).

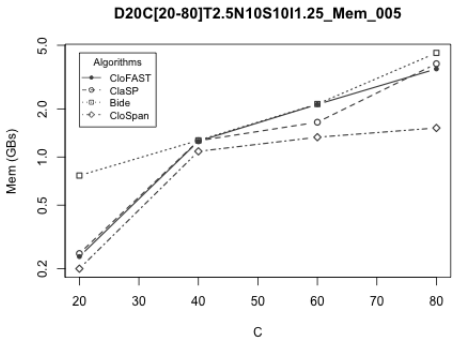
The results shown in Figure 5.14, indicate that, by increasing the number of sequences, CloFAST significantly outperforms ClaSP and BIDE, both in terms of running time and in terms of memory consumption. The comparison between CloFAST and CloSpan reveals that CloFAST outperforms CloSpan (although the difference is not impressive) in terms of running time. As concerns memory consumption, CloFAST outperforms CloSpan only when the number of sequences is less than 150.000. This is not surprising, since the value of the density is not high ($T/N = 8$), and CloFAST is more effective when the density increases (see Figure 5.9).

5.7.5 Effectiveness of closure checking and pruning

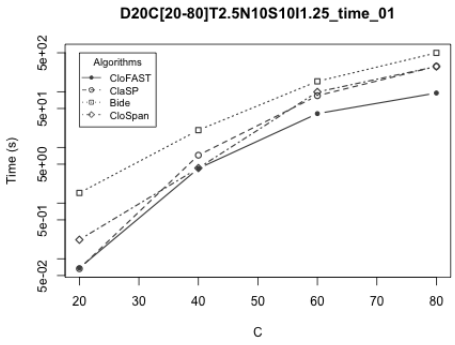
In this subsection Iinvestigate the effectiveness of the closure checking and of the pruning strategy implemented in CloFAST and when they come into play. To this aim, I consider three real-world datasets (i.e. Snake*, Pumbs and Pumbs*) and four synthetic datasets with different characteristics . I compare the results



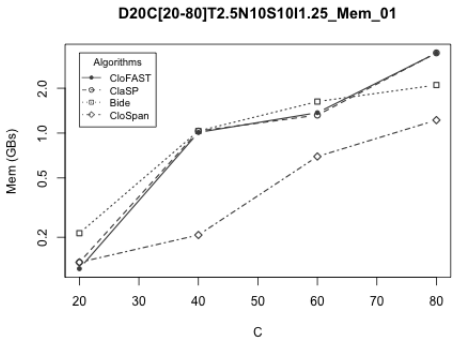
(a) D20C[20-80]T2.5N10S10I1.25 $min\_supp = 0.05$



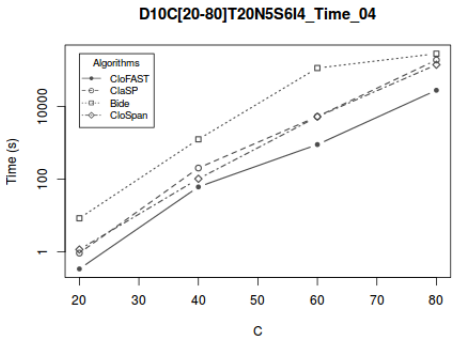
(d) D20C[20-80]T2.5N10S10I1.25 $min\_supp = 0.05$



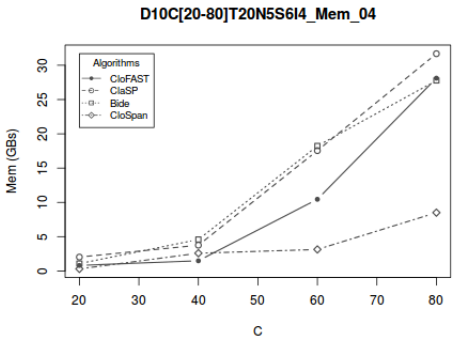
(b) D20C[20-80]T2.5N10S10I1.25 $min\_supp = 0.1$



(e) D20C[20-80]T2.5N10S10I1.25 $min\_supp = 0.1$



(c) D10C[20-80]T20N5S6I4 $min\_supp = 0.4$



(f) D10C[20-80]T20N5S6I4 $min\_supp = 0.4$

Figure 5.10: Running times (in seconds) and memory consumption (in Gb) varying $C = \{20, 40, 60, 80\}$. Results are obtained with $D = 20, min\_sup = 0.05, T = 2.5$ (sparse); $D = 20, min\_sup = 0.1, T = 2.5$ (sparse); $D = 10, min\_sup = 0.4, T = 20$ (dense).

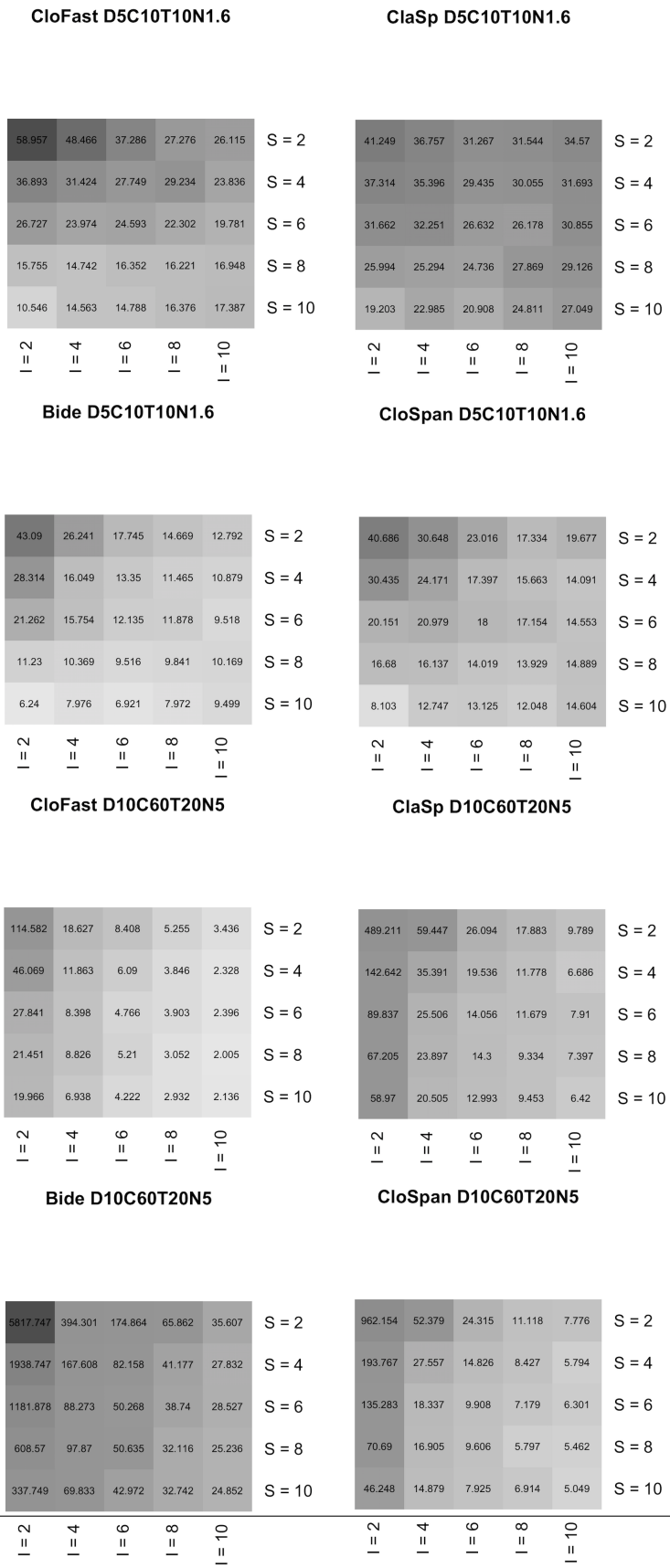
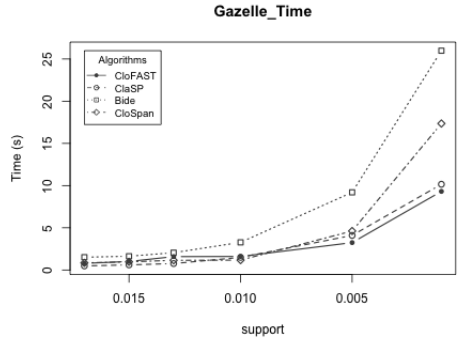
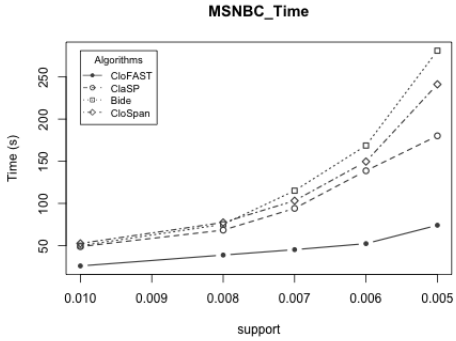


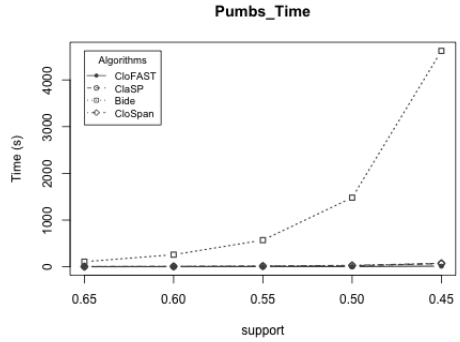
Figure 5.11: Running times (in seconds) varying $S = \{2, 4, 6, 8, 10\}$ and $I = \{2, 4, 6, 8, 10\}$. Results are obtained with $D = 5, C = 10, T = 10, N = 1.6, min\_sup = 0.05$ (small and sparse); $D = 10, C = 60, T = 20, N = 5, min\_sup = 0.7$ (dense).



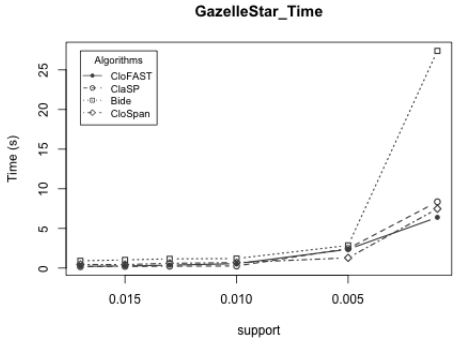
(a) Gazelle



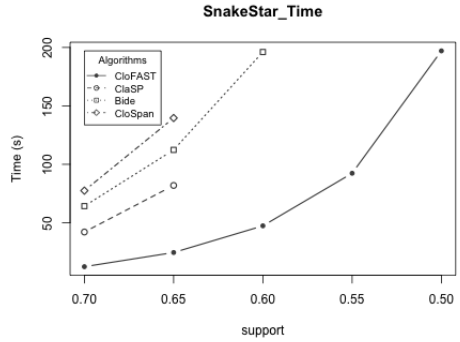
(b) MSNBC



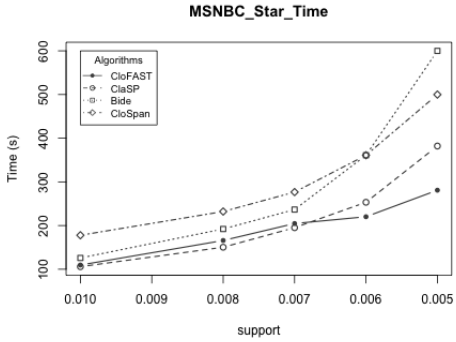
(c) Pumsb



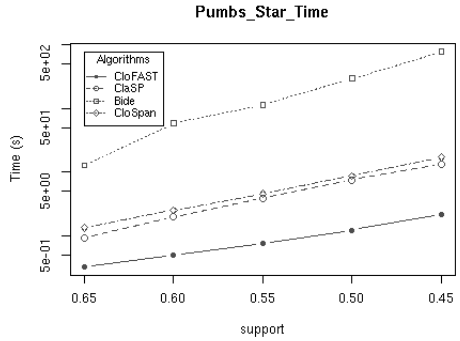
(d) Gazelle*



(e) Snake*

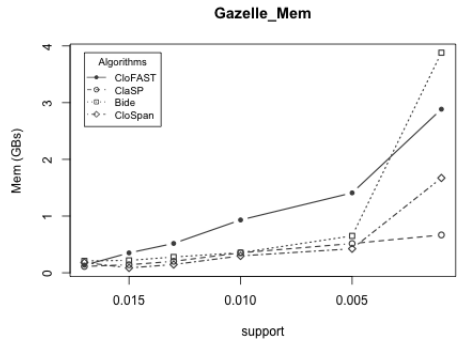


(f) MSNBC*

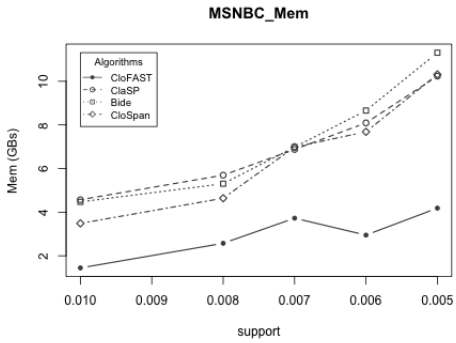


(g) Pumbs* (logarithmic scale)

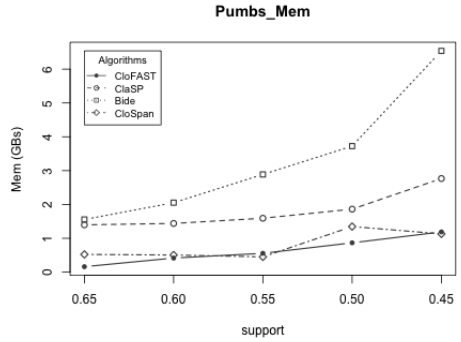
Figure 5.12: Real datasets: running times. Missing values correspond to out-of-memory errors (> 32GB).



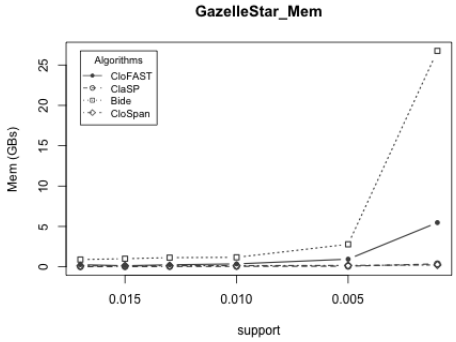
(a) Gazelle



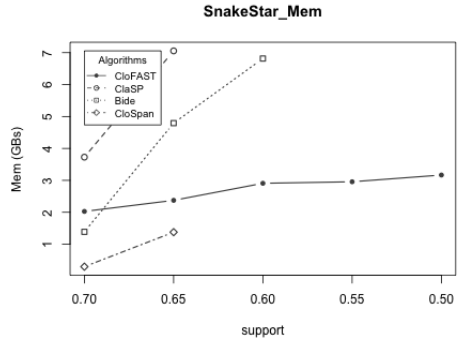
(b) MSNBC



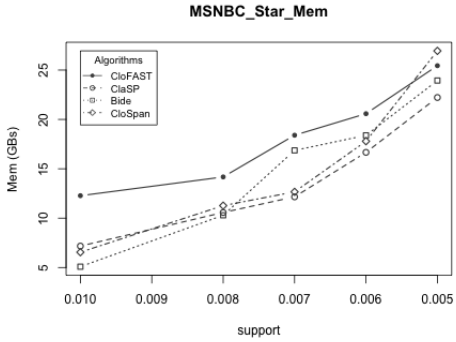
(c) Pumsb



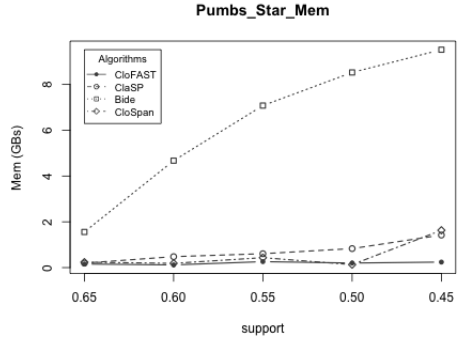
(d) Gazelle*



(e) Snake*



(f) MSNBC*



(g) Pumbs*

Figure 5.13: Real datasets: memory consumption. Missing values correspond to out-of-memory errors (> 32GB).

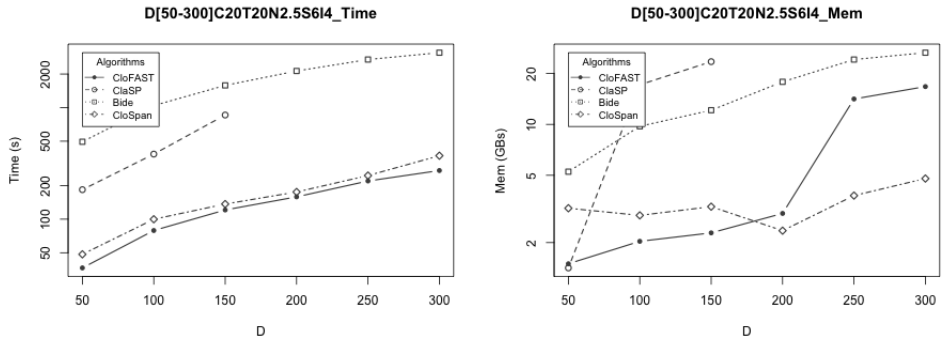


Figure 5.14: Running times (in seconds) and memory consumption (in Gb) varying D=50, 100, 150, 200, 250, 300. Results are obtained with C=20, T=20, N=2.5 and $min\_sup = 0.4$. Missing values correspond to out-of-memory errors (> 32GB).

of CloFAST with those of FAST [SFMH11], which does not perform closure checking and pruning.

The results reported in Figure 5.16 confirm, as expected, that CloFAST is able to prune a higher percentage of frequent sequences when input sequences are long and when the number of input sequences increases (Snake* vs. Pumbs/Pumbs*). By comparing the results obtained with Pumbs and Pumbs*, we notice that the benefits of CloFAST are more evident when the number of input sequences increases (the main difference between Pumbs and Pumbs* is in the number of sequences). Obviously, the percentage of pruned frequent sequences directly reflects on the running time and on the memory consumption. In particular, when the difference between the number of closed patterns and the number of frequent patterns increases, CloFAST shows a proportional improvement in terms of both running times and memory consumption. This is more evident for small values of the support threshold.

Experiments on synthetic datasets aimed at evaluating how the closure check and pruning performs when the number of frequent sequences in the underlying model changes. In particular, by keeping unchanged values of D, C, T and N, we can compare the results obtained with different values of S and I, which regulate the average length of maximal sequences and the average size of itemsets in maximal sequences, respectively. When S is small and I is large, the underlying patterns are very similar to each other and multiple sequences covered by the same pattern are likely to be generated, thus leading to better opportunities for pruning. The results (see Figure 5.17) show that with a small value of S

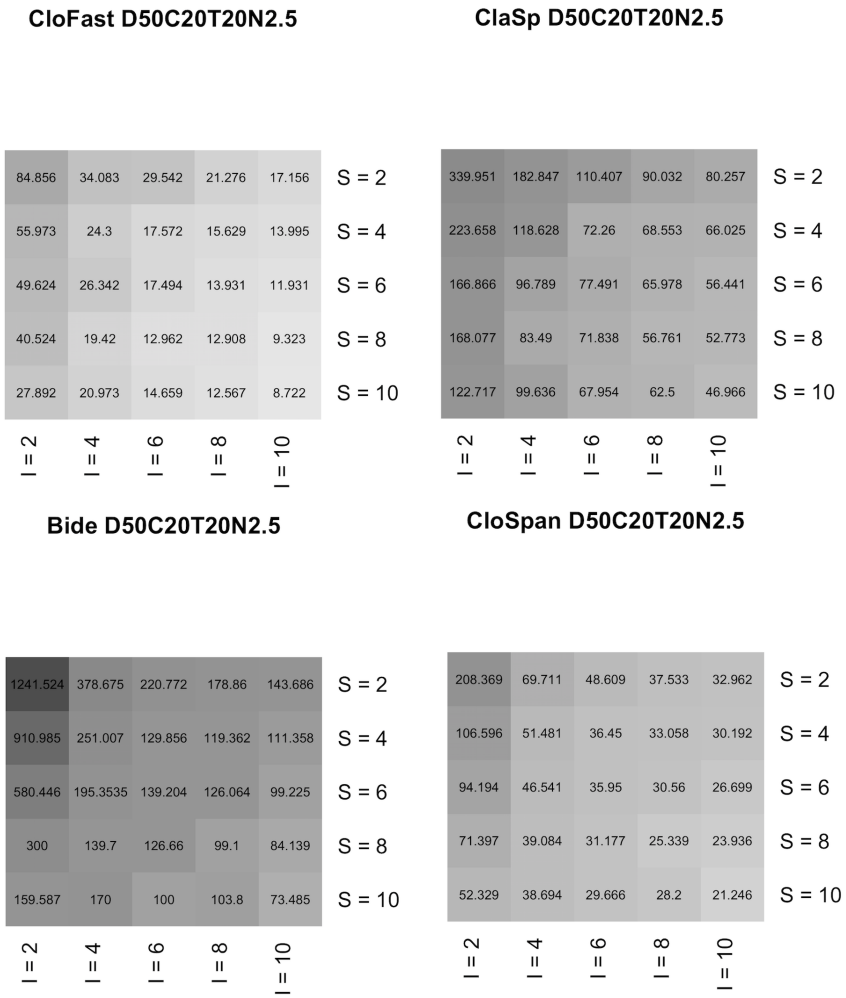


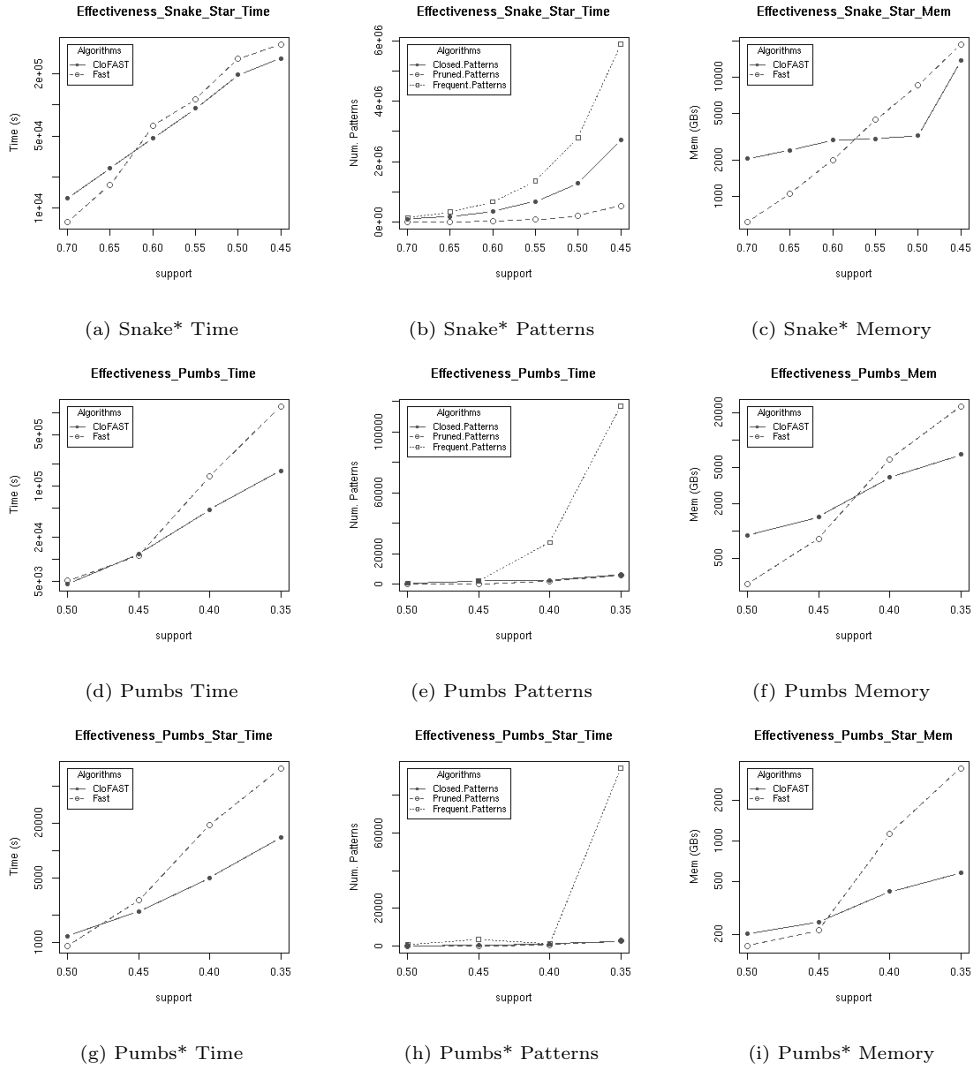
Figure 5.15: Running times (in seconds) varying $S = \{2, 4, 6, 8, 10\}$ and $I = \{2, 4, 6, 8, 10\}$. Results are obtained with $D = 50, C = 20, T = 20, N = 2.5, min\_sup = 0.4$.

and high value of I, CloFAST is able to prune the search space significantly, thus greatly outperforming FAST in terms of running times, at minor additional memory costs.

5.8 Conclusions

In this chapter I have presented CloFAST, a novel algorithm for mining closed frequent sequences without candidate maintenance. It exploits *i) sparse id-lists* and *vertical id-lists* for fast counting the support of sequential patterns and *ii)* a novel one-step technique to check *sequence closure* and to *prune* the search space. In this way, CloFAST is also able to mine long closed sequences by reducing the effort required for space exploration, support counting and search space pruning.

A thorough experimental study with both artificial and real datasets shows that CloFAST outperforms in running times the state-of-the-art algorithms, especially for low support values and for dense datasets, i.e. when the number of frequent sequences is high. Notably, the time efficiency of CloFAST is not obtained at the cost of higher memory consumption, which remains comparable to, if not better than, that of other systems (e.g. CloSPAN). For some critical datasets, CloFAST is the only system that returns a result within the fixed memory size constraints. A comparison with its predecessor FAST, which does not perform closure checking and pruning, shows that CloFAST is more efficient both in time and memory consumption, thus providing a way to compact results, while preserving the same expressive power of discovered patterns.



(a) Snake* Time (b) Snake* Patterns (c) Snake* Memory

(d) Pumbs Time (e) Pumbs Patterns (f) Pumbs Memory

(g) Pumbs* Time (h) Pumbs* Patterns (i) Pumbs* Memory

Figure 5.16: Fast and CloFast comparison on real datasets

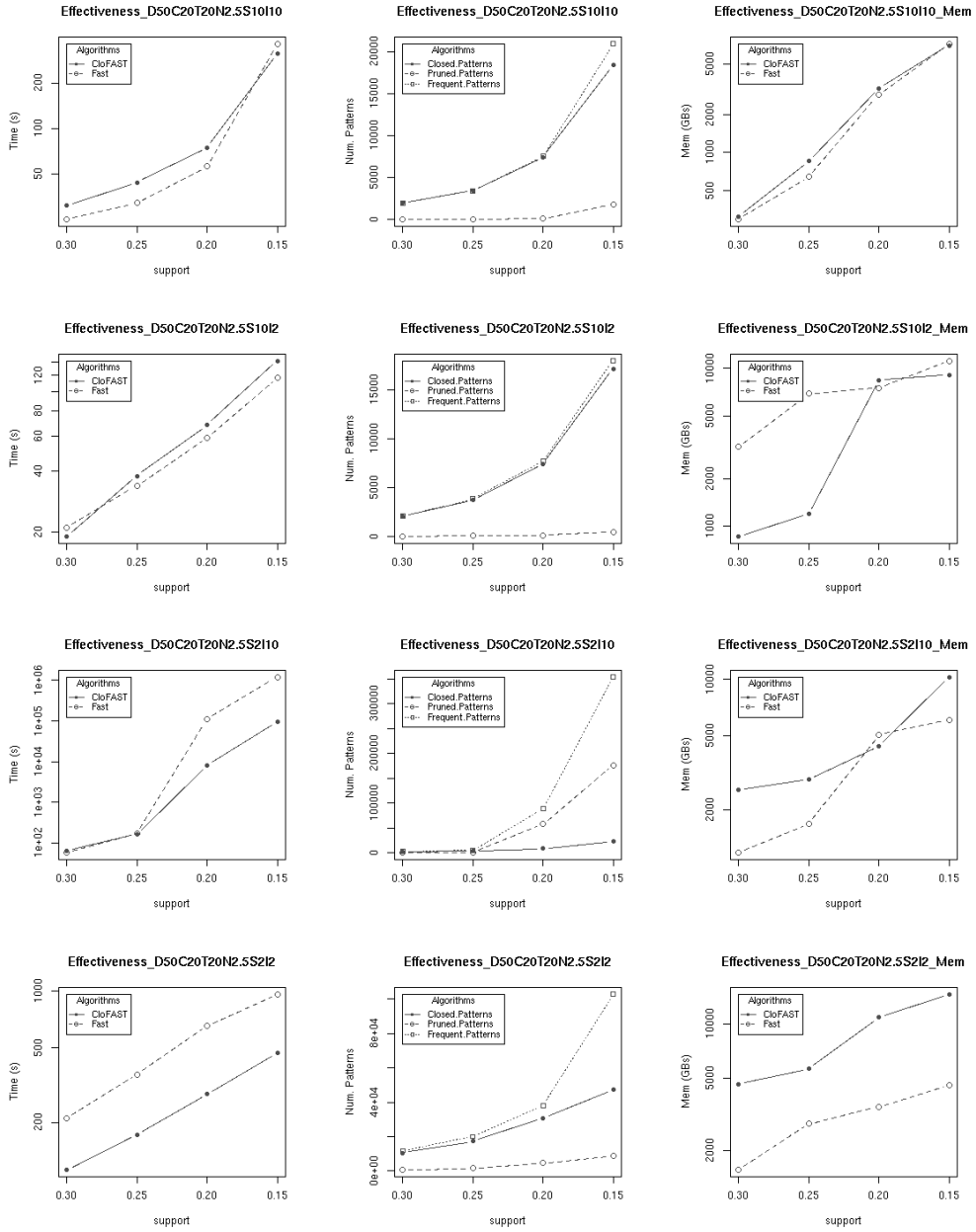


Figure 5.17: Fast and CloFast comparison on synthetic datasets.

6

Conclusion

The World Wide Web, in its current form, is modern legacy system where data cannot be automatically accessed and manipulated. Goal of Web Mining is therefore to discover and to extract valuable information in the way to improve and automate the information gathering process.

In this thesis I face the problem of mining structured data encoded in web pages or hidden in the topological structure of a website. This is due analyzing the most important representations of a web page (i.e. textual, visual, structural representation) and combining the extracted intra-page information with inter-page information (i.e. hyperlink structure). In particular this thesis focus on two main open challenges, that is combining structured data splitted on multiple web pages and organizing web pages semantically similar.

In the first case, similar to databases, where a view can represent a subset of the data contained in a table, websites split a logical list in multiple views in order to avoid information overload and to facilitate users' navigation. However, the data stored in such *logical list* need to be automatically extracted to enable building services for market intelligence, synonyms discovery, question answering and data mashup. The approach presented in this thesis solves this open issue. Experimental results show that the algorithm is extremely accurate and it is able to extract *logical lists* in a wide range of domains and websites with high precision and recall. Part of this future work will involve tasks such as indexing the Web based on lists and tables, answering queries from lists, and entity discovery and disambiguation using lists.

In the second case, web pages can be grouped based on different types of features. To solve this open issue, I present two unsupervised and domain-independent approaches. In particular, the first method combines information about content, web page structure and hyperlink structure in a single vector

space representation which can be used by any traditional and best-performing clustering algorithms. To take into account the hyperlink structure I exploit recent advances in natural language processing by adapting the skip-gram model. Experiments results show that content and hyperlink structure of web pages provide different and complementary information which can improve the efficacy of clustering algorithms. Moreover, experiments do not show statistical differences between results which use web lists and results obtained ignoring web page structure. The second approach proposed in this thesis is related with the automatic sitemap generation. A sitemap represents an explicit specification of the design concept and knowledge organization of a website. Intuitively, sibling Web pages this hierarchy should be of the same type (e.g. professors web pages, courses web pages, etc.), and parents and children pages should be of more general and more specific types respectively. For this purpose I propose a new method which automatically discovers the hierarchical organization of a website by using the content and the hyperlinks structure of web pages. Experimental results prove that the proposed approach outperforms HDTM, a state-of-art algorithm which extract the hierarchical structure of a website through the analysis of contents and hyperlinks

Bibliography

[ADW01] Corin R. Anderson, Pedro Domingos, and Daniel S. Weld. Adaptive web navigation for wireless devices. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, pages 879–884, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[AFGY02] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 429–435, New York, NY, USA, 2002. ACM.

[AGM02] Arvind Arasu and Hector Garcia-Molina. Extracting structured data from web pages. Technical Report 2002-40, Stanford InfoLab, July 2002.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[AS06] Ralitsa Angelova and Stefan Siersdorfer. A neighborhood-based approach for clustering of linked document collections. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 778–779, New York, NY, USA, 2006. ACM.

[AWP14] Basavaraj S. Anami, Ramesh S. Wadawadagi, and Veerappa B. Pagi. Machine learning techniques in web content mining: A comparative analysis. *Journal of Information & Knowledge Management (JIKM)*, 13(01), 2014.

[AZ12] Charu C. Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 77–128. Springer, 2012.

[BBA+00] Matthias Baumgarten, Alex G. Büchner, Sarabjot S. Anand, Maurice D. Mulvenna, and John G. Hughes. User-driven navigation pattern discovery from internet data. In *Revised Papers from the International Workshop on Web Usage Analysis and*

User Profiling, WEBKDD '99, pages 74–91, London, UK, UK, 2000. Springer-Verlag.

[BCF+05] Doug Burdick, Manuel Calimlim, Jason Flannick, Johannes Gehrke, and Tomi Yiu. MAFIA: A maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1490–1504, 2005.

[BCMP13] Mirko Bronzi, Valter Crescenzi, Paolo Merialdo, and Paolo Papotti. Extraction and integration of partially overlapping web sources. *Proc. VLDB Endow.*, 6(10):805–816, August 2013.

[BDM11] Lorenzo Blanco, Nilesh Dalvi, and Ashwin Machanavajjhala. Highly efficient algorithms for structural clustering of large websites. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 437–446, New York, NY, USA, 2011. ACM.

[BG10] Paul Bohunsky and Wolfgang Gatterbauer. Visual structure-based web page clustering and retrieval. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1067–1068, New York, NY, USA, 2010. ACM.

[BP12] Sergey Brin and Lawrence Page. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 56(18):3825–3833, 2012.

[But04] David Buttler. A short survey of document structure similarity algorithms. In *Proceedings of the International Conference on Internet Computing, IC '04, Las Vegas, Nevada, USA, June 21-24, 2004, Volume 1*, pages 3–9, 2004.

[CAC08] Morteza Haghir Chehreghani, Hassan Abolhassani, and Mostafa Haghir Chehreghani. Improving density-based methods for hierarchical clustering of web pages. *Data Knowl. Eng.*, 67(1):30–50, October 2008.

[CCdM+03] Marco Cristo, Pável Calado, Edleno Silva de Moura, Nivio Ziviani, and Berthier A. Ribeiro-Neto. Link information as a similarity measure in web classification. In *String Processing and Information Retrieval, 10th International Symposium, SPIRE 2003, Manaus, Brazil, October 8-10, 2003, Proceedings*, pages 43–55, 2003.

[CCM+03] Pável Calado, Marco Cristo, Edleno Moura, Nivio Ziviani, Berthier Ribeiro-Neto, and Marcos André Gonçalves. Combining link-based and content-based methods for web document classification. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, CIKM '03, pages 394–401, New York, NY, USA, 2003. ACM.

[Che08] Max Chevalier. Zdravko markov and daniel t. larose, data mining the web: Uncovering patterns in web content, structure, and usage. *Inf. Retr.*, 11(2):169–174, 2008.

[CKGS06] Chia-Hui Chang, Mohammed Kayed, Moheb Ramzy Girgis, and Khaled F. Shaalan. A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng.*, 18(10):1411–1428, October 2006.

[CMM05] Valter Crescenzi, Paolo Merialdo, and Paolo Missier. Clustering web pages based on their structure. *Data Knowl. Eng.*, 54(3):279–299, September 2005.

[CMS13] Ricardo J. G. B. Campello, Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. *Advances in Knowledge Discovery and Data Mining*, 2013.

[CWYM06] Yun Chi, Haixun Wang, Philip S. Yu, and Richard R. Muntz. Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowledge and Information Systems*, 10:265–294, October 2006.

[DBS05] Isabel Drost, Steffen Bickel, and Tobias Scheffer. Discovering communities in linked data by multi-view clustering. In Myra Spiliopoulou, Rudolf Kruse, Christian Borgelt, Andreas NÃ$\frac{1}{4}$rnberger, and Wolfgang Gaul, editors, *GfKl*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 342–349. Springer, 2005.

[DGH+14] Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Kevin Murphy, Shaohua Sun, and Wei Zhang. From data fusion to knowledge fusion. *Proc. VLDB Endow.*, 7(10):881–892, June 2014.

[DMP12] Nilesh Dalvi, Ashwin Machanavajjhala, and Bo Pang. An analysis of structured data on the web. *Proc. VLDB Endow.*, 5(7):680–691, March 2012.

[DMRW09] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An optimal decomposition algorithm for tree edit distance. *ACM Trans. Algorithms*, 6(1):2:1–2:19, December 2009.

[Eik99] Line Eikvil. Information extraction from world wide web - a survey, 1999.

[ETPF08] Themis P. Exarchos, Markos G. Tsipouras, Costas Papaloukas, and Dimitrios I. Fotiadis. A two-stage methodology for sequence classification based on sequential pattern mining and optimization. *Data & Knowledge Engineering*, 66:467–487, September 2008.

[FAN04] Mohamed Fathi, N Adly, and M Nagi. Web documents classification using text, anchor, title and metadata information. In *Proceedings of the international conference on computer science, software engineering, information technology, e-Business and Applications*, pages 1–8, 2004.

[Fan07] 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.

[FE03] Michelle Fisher and Richard Everson. When are links useful? experiments in text classification. In *Proceedings of the 25th European Conference on IR Research*, ECIR'03, pages 41–56, Berlin, Heidelberg, 2003. Springer-Verlag.

[Fir57] J. R. Firth. A synopsis of linguistic theory 1930-55. 1952-59:1–32, 1957.

[FLCM16] Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. Clofast: closed sequential pattern mining using sparse and vertical id-lists. *Knowledge and Information Systems*, 2016.

[FM10] Dmitriy Fradkin and Fabian Moerchen. Margin-closed frequent sequential pattern mining. In *Proceedings of the ACM SIGKDD Workshop on Useful Patterns*, UP '10, pages 45–54, New York, NY, USA, 2010. ACM.

[FMFB12] Emilio Ferrara, Pasquale De Meo, Giacomo Fiumara, and Robert Baumgartner. Web data extraction, applications and techniques: A survey. *CoRR*, abs/1207.0246, 2012.

[FV] Philippe Fournier-Viger. SPMF: A sequential pattern mining framework. `http://www.philippe-fournier-viger.com/spmf/index.php`. Accessed: 2014-08-08.

[GBH+07] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 71–80, New York, NY, USA, 2007. ACM.

[GCMG13] Antonio Gomariz, Manuel Campos, Roque Marín, and Bart Goethals. ClaSP: An efficient algorithm for mining frequent closed sequences. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *PAKDD (1)*, volume 7818 of *Lecture Notes in Computer Science*, pages 50–61. Springer, 2013.

[GGV15] Olof Gornerup, Daniel Gillblad, and Theodore Vasiloudis. Knowing an object by the company it keeps: A domain-agnostic scheme for similarity discovery. In *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*, ICDM '15, pages 121–130, Washington, DC, USA, 2015. IEEE Computer Society.

[GHFZ13] Adrien Guille, Hakim Hacid, Cecile Favre, and Djamel A. Zighed. Information diffusion in online social networks: A survey. *SIGMOD Rec.*, 42(2):17–28, July 2013.

[Got08] Thomas Gottron. *Clustering Template Based Web Documents*, pages 40–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[GTL+02] Eric J. Glover, Kostas Tsioutsiouliklis, Steve Lawrence, David M. Pennock, and Gary W. Flake. Using web structure for classifying and describing web pages. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 562–569, New York, NY, USA, 2002. ACM.

[HA85] L. Hubert and P. Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[HAB12] Sven Helmer, Nikolaus Augsten, and Michael Böhlen. Measuring structural similarity of semistructured data based on information-theoretic approaches. *The VLDB Journal*, 21(5):677–702, October 2012.

[Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[HCTH06] Kuo-Yu Huang, Chia-Hui Chang, Jiun-Hung Tung, and Cheng-Tao Ho. COBRA: Closed sequential pattern mining using bi-phase reduction approach. In A. Min Tjoa and Juan Trujillo, editors, *DaWaK*, volume 4081 of *Lecture Notes in Computer Science*, pages 280–291. Springer, 2006.

[HGKI02] Taher H. Haveliwala, Aristides Gionis, Dan Klein, and Piotr Indyk. Evaluating strategies for similarity search on the web. In *Proceedings of the 11th International Conference on World Wide Web*, WWW '02, pages 432–442, New York, NY, USA, 2002. ACM.

[HMT11] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, May 2011.

[HNVV03] Maria Halkidi, Benjamin Nguyen, Iraklis Varlamis, and Michalis Vazirgiannis. Thesus: Organizing web document collections based on link semantics. *The VLDB Journal*, 12(4):320–332, November 2003.

[HZHDDS02] Xiaofeng He, Hongyuan Zha, Chris H.Q. Ding, and Horst D. Simon. Web document clustering using hyperlink structures. *Comput. Stat. Data Anal.*, 41(1):19–45, November 2002.

[Jai10] Anil K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recogn. Lett.*, 31(8):651–666, June 2010.

[JZ10] Guozhu Gao Jingjun Zhu, Haiyan Wu. An efficient method of web sequential pattern mining based on session filter and transaction identification. *Journal of Networks*, 5(9):1017–1024, 2010.

[Kal07] James Kalbach. *Designing Web Navigation*. O'Reilly, first edition, 2007.

[KC12] Myung Hee Kim and Paul Compton. Improving the performance of a named entity recognition system with knowledge acquisition. In *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management*, EKAW'12, pages 97–113, Berlin, Heidelberg, 2012. Springer-Verlag.

[Kes63] M. M. Kessler. Bibliographic coupling between scientific papers. *American Documentation*, 14:10–25, 1963.

[KHY+08] Hillol Kargupta, Jiawei Han, Philip S. Yu, Rajeev Motwani, and Vipin Kumar. *Next Generation of Data Mining*. Chapman & Hall/CRC, 1 edition, 2008.

[Kle99] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999.

[KMH13] Matthias Keller, Patrick Mühlschlegel, and Hannes Hartenstein. 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, pages 1269–1274, Republic and Canton of Geneva, Switzerland, 2013. International World Wide Web Conferences Steering Committee.

[KN12] Matthias Keller and Martin 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*, WWW '12 Companion, pages 1025–1034, New York, NY, USA, 2012. ACM.

[Kus97] Nicholas Kushmerick. Wrapper induction for information extraction. In *Proc. IJCAI-97*, 1997.

[LB99] Hakon Wium Lie and Bert Bos. *Cascading Style Sheets:Designing for the Web, 2nd Edition*. Addison-Wesley Professional, 1999.

[LCC11] Shian-Hua Lin, Kuan-Pak Chu, and Chun-Ming Chiu. Automatic sitemaps generation: Exploring website structures using block extraction and hyperlink analysis. *Expert Systems with Applications*, 38(4):3944–3958, 2011.

[Lev10] Mark Levene. *An Introduction to Search Engines and Web Navigation*. Wiley Publishing, 2nd edition, 2010.

[LFC+14] Pasqua Fabiana Lanotte, Fabio Fumarola, Michelangelo Ceci, Andrea Scarpino, MicheleDamiano Torelli, and Donato Malerba. Automatic extraction of logical web lists. In Troels Andreasen, Henning Christiansen, Juan-Carlos Cubero, and ZbigniewW. RaÅ›, editors, *Foundations of Intelligent Systems*, volume 8502 of *Lecture Notes in Computer Science*, pages 365–374. Springer International Publishing, 2014.

[LGMK04] Kristina Lerman, Lise Getoor, Steven Minton, and Craig Knoblock. Using the structure of web sites for automatic segmentation of tables. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 119–130, New York, NY, USA, 2004. ACM.

[LGZ04] Bing Liu, Robert L. Grossman, and Yanhong Zhai. Mining web pages for data records. *IEEE Intelligent Systems*, 19(6):49–55, 2004.

[Liu06] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[LLC05] Uichin Lee, Zhenyu Liu, and Junghoo Cho. Automatic identification of user goals in web search. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 391–400, New York, NY, USA, 2005. ACM.

[LLMZ06] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, 32:176–192, 2006.

[LMM10] Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447–460, 2010.

[LNL04] Zehua Liu, Wee Keong Ng, and Ee-Peng Lim. An automated algorithm for extracting website skeleton. In *Database Systems for Advanced Applications*, pages 799–811. Springer, 2004.

[LYHL10] Cindy Xide Lin, Yintao Yu, Jiawei Han, and Bing Liu. Hierarchical web-page clustering via in-page and cross-page link struc-

tures. In *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'10, pages 222–229, Berlin, Heidelberg, 2010. Springer-Verlag.

[LZW+10] Cindy Xide Lin, Bo Zhao, Tim Weninger, Jiawei Han, and Bing Liu. Entity relation discovery from web tables and links. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1145–1146, New York, NY, USA, 2010. ACM.

[Mob07] Bamshad Mobasher. The adaptive web. chapter Data Mining for Web Personalization, pages 90–135. Springer-Verlag, Berlin, Heidelberg, 2007.

[MPT09] F. Masseglia, P. Poncelet, and M. Teisseire. Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Systems with Applications: An International Journal*, 36:2677–2690, March 2009.

[MR13] Carl H. Mooney and John F. Roddick. Sequential pattern mining – approaches and algorithms. *ACM Comput. Surv.*, 45(2):19:1–19:39, March 2013.

[MS00] Dharmendra S. Modha and W. Scott Spangler. Clustering hypertext with applications to web searching. In *Proceedings of the Eleventh ACM on Hypertext and Hypermedia*, HYPERTEXT '00, pages 143–152, New York, NY, USA, 2000. ACM.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[MTH09] Gengxin Miao, J Tatemura, and WP Hsiung. Extracting data records from the web using tag path clustering. *The World Wide Web Conference*, pages 981–990, 2009.

[NL06] Jakob Nielsen and Hoa Loranger. *Prioritizing Web Usability*. New Riders Publishing, Thousand Oaks, CA, USA, 2006.

[NM03] Miki Nakagawa and Bamshad Mobasher. A hybrid web personalization model based on site connectivity. In *Proceedings of WebKDD*, pages 59–70, 2003.

[OC03] Christopher Olston and Ed H. Chi. Scenttrails: Integrating browsing and searching on the web. *ACM Trans. Comput.-Hum. Interact.*, 10(3):177–197, September 2003.

[PARS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 701–710, New York, NY, USA, 2014. ACM.

[PHMA+01] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. PrefixSpan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering*, pages 215–224, Washington, DC, USA, 2001. IEEE Computer Society.

[PL05] P. Pons and M. Latapy. Computing communities in large networks using random walks (long version). *ArXiv Physics e-prints*, December 2005.

[PLCM14] Gianvito Pio, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. Mining temporal evolution of entities in a stream of textual documents. In *Foundations of Intelligent Systems - 21st International Symposium, ISMIS 2014, Roskilde, Denmark, June 25-27, 2014. Proceedings*, pages 50–60, 2014.

[PPLM06] Pons, Pascal, Latapy, and Matthieu. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006.

[QD06] Xiaoguang Qi and Brian D. Davison. Knowing a web page by the company it keeps. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 228–237, New York, NY, USA, 2006. ACM.

[QD09] Xiaoguang Qi and Brian D. Davison. Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(2):12:1–12:31, February 2009.

[RH07] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in NLP and Computational Natural Language Learning(EMNLP-CoNLL)*, pages 410–420, 2007.

[Sah08] Magnus Sahlgren. The distributional hypothesis. *Italian Journal of Linguistics*, 20(1):33–54, 2008.

[Sar08] Sunita Sarawagi. Information extraction. *Found. Trends databases*, 1(3):261–377, March 2008.

[SCDT00] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl.*, 1(2):12–23, January 2000.

[SFMH11] Eliana Salvemini, Fabio Fumarola, Donato Malerba, and Jiawei Han. FAST sequence mining based on sparse id-lists. In Marzena Kryszkiewicz, Henryk Rybinski, Andrzej Skowron, and Zbigniew W. Ras, editors, *ISMIS*, volume 6804 of *Lecture Notes in Computer Science*, pages 316–325. Springer, 2011.

[SH12] Yizhou Sun and Jiawei Han. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Morgan & Claypool Publishers, 2012.

[SHJ05] Shijie Song, Huaping Hu, and Shiyao Jin. HVSM: A new sequential pattern mining algorithm using bitmap representation. In Xue Li, Shuliang Wang, and ZhaoYang Dong, editors, *Advanced Data Mining and Applications*, volume 3584 of *Lecture Notes in Computer Science*, pages 455–463. Springer Berlin Heidelberg, 2005.

[Sma73] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24(4):265–269, 1973.

[TLS+09] Antonio Turi, Corrado Loglisci, Eliana Salvemini, Giorgio Grillo, Donato Malerba, and Domenica D'Elia. Computational annotation of UTR cis-regulatory modules through frequent pattern mining. *BMC Bioinformatics*, 10:1–12, 2009. 10.1186/1471-2105-10-S6-S25.

[TP10] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *J. Artif. Int. Res.*, 37(1):141–188, January 2010.

[TQW+15] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 1067–1077, New York, NY, USA, 2015. ACM.

[WBH12] Tim Weninger, Yonatan Bisk, and Jiawei Han. Document-topic hierarchies from document graphs. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 635–644, New York, NY, USA, 2012. ACM.

[WHL07] Jianyong Wang, Jiawei Han, and Chun Li. Frequent closed sequence mining without candidate maintenance. *IEEE Trans. on Knowl. and Data Eng.*, 19:1042–1056, August 2007.

[WJH13] Tim Weninger, Thomas J. Johnston, and Jiawei Han. The parallel path framework for entity discovery on the web. *ACM Trans. Web*, 7(3):16:1–16:29, September 2013.

[WK02] Yitong Wang and Masaru Kitsuregawa. Evaluating contents-link coupled web page clustering for web search results. In *Proceedings of the Eleventh International Conference on Information and Knowledge Management*, CIKM '02, pages 499–506, New York, NY, USA, 2002. ACM.

[YB07] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1091–1095, June 2007.

[YHA03] Xifeng Yan, Jiawei Han, and Ramin Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *In SDM*, pages 166–177, 2003.

[YK05] Zhenglu Yang and Masaru Kitsuregawa. LAPIN-SPAM: An improved algorithm for mining sequential pattern. *Data Engineering Workshops, 22nd International Conference on*, 0:1222, 2005.

[YL09] Christopher C. Yang and Nan Liu. Web site topic-hierarchy generation based on link structure. *J. Am. Soc. Inf. Sci. Technol.*, 60(3):495–508, March 2009.

[YLL03] Lan Yi, Bing Liu, and Xiaoli Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 296–305, New York, NY, USA, 2003. ACM.

[Zak01] Mohammed J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, January 2001.

[ZCY10] Yang Zhou, Hong Cheng, and Jeffrey Xu Yu. Clustering large attributed graphs: An efficient incremental approach. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 689–698, Washington, DC, USA, 2010. IEEE Computer Society.

[ZDR00] Xiuzhen Zhang, Guozhu Dong, and Kotagiri Ramamohanarao. Exploring constraints to efficiently mine emerging patterns from large high-dimensional datasets. In *Knowledge Discovery and Data Mining*, pages 310–314, 2000.

[ZE98] Oren Zamir and Oren Etzioni. Web document clustering: A feasibility demonstration. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '98, pages 46–54, New York, NY, USA, 1998. ACM.

[ZL05] Yanhong Zhai and Bing Liu. Web data extraction based on partial tree alignment. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pages 76–85, New York, NY, USA, 2005. ACM.

[ZYCG07] Shenghuo Zhu, Kai Yu, Yun Chi, and Yihong Gong. Combining content and link for classification using matrix factorization. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 487–494, New York, NY, USA, 2007. ACM.