

Information Systems Research Group
Department of Informatics
University of Fribourg



Harnessing Folksonomies with a Web Crawler

Bachelor Thesis

Author:

David Oggier
david.oggier@unifr.ch
Ch. Des Grenadiers 2
1700 Fribourg

Assistance:

Edy Portmann

Supervision:

Prof. Dr. Andreas Meier

December 2009

Abstract

Folksonomies play a central role in the current trend of the World Wide Web design towards collaboration and information sharing. They provide a way to create metainformation in a simple, yet precise manner with little additional costs. The contribution of a large number of users creates a network effect that allow for folksonomies to annotate and categorize a vast amount of resources while including different vocabularies. Various resource sharing and social bookmarking platforms encourage folksonomy tagging. The resulting metainformation, however, can only be queried by the respective platform's search tool. But since all the folksonomy data are publicly available, they could potentially be harnessed by outside programs, such as Web crawlers, in order to take advantage of the knowledge stored in them to better index the corresponding page. This is precisely the central subject of this paper, in which a few ways to do so are explored and analyzed and later on one specific design is implemented with a prototype.

In an initial theoretical part, the subjects of Web search engines, and more thoroughly Web crawlers, are discussed, as well as folksonomies and the related subject of the Semantic Web. A practical part then introduces the problem to be addressed by this paper and proposes solutions to it. Finally, a simple Web crawler prototype will be described and analyzed.

Keywords: Folksonomy, Web crawler, search engine, metadata, Semantic Web.

Table of Contents

Abstract	I
Table of Contents	II
List of Figures	V
List of Tables	VI
List of Acronyms	VII
1 Introduction	1
1.1 Problem Description	1
1.2 Objectives	2
1.3 Methodology	3
<i>Part I: Introducing the Theoretical Concepts</i>	4
2 The Web Search Engine	5
2.1 Introduction	5
2.2 A Brief History of Web Search Engines	5
2.3 The Architecture of Web Search Engines	6
2.3.1 A Brief Introduction to Web Crawlers	7
2.3.2 Indexing	7
2.3.3 The Web Search Query and Link Analysis	9
2.3.4 Query Categorization	11
2.4 The Web Crawler	12
2.4.1 Fetching and Parsing	12
2.4.2 The Frontier	14
2.4.3 Additional Crawler Functions	15
2.4.4 Crawler Etiquette	16
2.4.5 Crawling Algorithms	17

2.4.6	Web Crawler Challenges	18
3	The Semantic Web and Folksonomies	21
3.1	Introduction	21
3.2	Metadata	21
3.2.1	A Classification of Metadata	21
3.2.2	Benefits and Limitations of Metadata	22
3.3	The Semantic Web and Ontologies	23
3.3.1	The Purpose of the Semantic Web	24
3.3.2	Semantic Web Standards	25
3.3.3	Ontologies in the Semantic Web	27
3.3.4	The Future of the Semantic Web	30
3.4	Folksonomies	31
3.4.1	The Purpose of Folksonomies	31
3.4.2	Strengths and Weaknesses of Folksonomies	32
3.4.3	Folksonomies in Usage	33
	<i>Part II: Harnessing Folksonomies in Theory and Practice</i>	36
4	Harnessing Folksonomies	37
4.1	Introduction	37
4.2	Folksonomies and Web Crawlers	37
4.2.1	Identification of Tags	37
4.2.2	The Use of Folksonomy Harnessing	38
4.3	Narrow and Broad Folksonomies in Practice	39
4.3.1	Definition of Folksonomy Revisited	39
4.3.2	Practical Example of a Narrow Folksonomy	40
4.3.3	Practical Example of a Broad Folksonomy	42
4.4	Methods for Harnessing Folksonomy Tags	45

4.4.1	Extraction of Formally Defined Tags	46
4.4.2	Harnessing the Folksonomies of Social Bookmarking Platforms	47
4.4.3	Ad-hoc Tag Extraction	48
4.4.4	Semi-Automatic Tag Identification	50
4.5	Indexing and Ranking with Folksonomy Tags	52
4.5.1	Parsing Folksonomy Tags	52
4.5.2	Indexing with Folksonomy Tags	53
4.5.3	Ranking Documents with Folksonomy Tags	54
5	A Web Crawler Prototype	57
5.1	Introduction	57
5.2	Crawler Design	57
5.2.1	The Prototype's Scope	57
5.2.2	Fetching and Crawling Web Pages	58
5.2.3	Web Page Parsing and Indexing	59
5.2.4	Graphical User Interface	60
5.2.5	Crawl Results and Search Functionality	62
5.3	Prototype Evaluation	63
5.3.1	Overall Performance	63
5.3.2	Numerical Evaluation	64
5.3.3	Future Extensions and Improvements	66
6	Conclusion	68
6.1	Paper Synopsis and Results	68
6.2	Outlook	69
	References	70
	Appendix A: Prototype User Manual	
	Appendix B: CD-ROM Resources	

List of Figures

Fig.2.1	A Web search engine architecture (own illustration)	7
Fig.2.2	A network of documents with their PageRank score [Markov & Larose 2007]	10
Fig.2.3	Flow of a basic sequential crawler [Pant et al. 2004]	13
Fig.2.4	A frontier implemented as a FIFO queue (own illustration)	14
Fig.2.5	A multithreaded crawler model [Pant et al. 2004]	20
Fig.3.1	An example RDF graph (own illustration)	25
Fig.3.2	The Semantic Web architecture [Obitko 2007]	26
Fig.3.3	Distribution curve of tag frequency (own illustration)	34
Fig.4.1	Screenshot of Flickr's most popular tags represented in a tag cloud [Flickr 2009]	41
Fig.4.2	Screenshot of the Delicious tag cloud [Delicious 2009]	44
Fig.5.1	The prototype's frame to add a tag page and start a crawl (own illustration)	61
Fig.5.2	Screenshot of the prototype's result frame (own illustration)	62

List of Tables

Tab.2.1	Simplified examples of a forward index and an inverted index (own illustration)	8
Tab.2.2	A robots.txt example (own illustration)	17
Tab.3.1	An example RDF/XML statement (own illustration)	26
Tab.3.2	An example class declaration with OWL (own illustration)	28
Tab.3.3	An example property definition with OWL (own illustration)	29
Tab.3.4	An example cardinality restriction on a property with OWL (own illustration)	29
Tab.3.5	An example <i>equivalentClass</i> declaration with OWL (own illustration)	30
Tab.4.1	Sample code for a Flickr resource [Flickr 2009 (2)]	49
Tab.4.2	Sample code of Delicious tags [Delicious 2009 (2)]	50
Tab.5.1	Results for crawls of different lengths on YouTube and Flickr (own illustration)	65

List of Acronyms

ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DL	Description Logic
DNS	Domain Name System
FAQ	Frequently Asked Questions
FIFO	First In First Out
FTP	File Transfer Protocol
GUI	Graphical User Interface
HITS	Hyperlink-Induced Topic Search
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
KB, MB	Kilobyte, Megabyte
OWL	Web Ontology Language
PDF	Portable Document Format
RDF	Resource Description Framework
RDFS	RDF Schema
SPARQL	SPARQL Protocol And RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

1 Introduction

1.1 Problem Description

A main focus on the World Wide Web has recently been on user participation. While formerly the Web consisted mainly of static pages created by people at least to some extent knowledgeable in Web design, today there exist almost no more boundaries as to how even inexperienced users can shape it. Texts, photographs or videos can easily be shared with a large number of people across the world without any technology-specific knowledge or experience. However, along with the rapid increase of available information on the Web, the difficulty of keeping an overview of this mountain of resources also grows.

Nowadays, the default method to retrieve a specific Web page or generally information on a topic is to query Web search engines. These are tools that process and index a great amount of pages in order to be able to return a set of links as closely matching the keywords specified by the user as possible. Search engines are very successful at finding pertinent Web pages on any given subject. In spite of that, in order to find precisely what a user is looking for, the set of keywords specified by him need to correspond to the one extracted from Web pages by the search engine, or at least overlap. This is due to the fact that traditional search engines don't use synonyms to enhance their set of potentially relevant pages¹. For example, a query using the term “movie” will usually not return Web pages indexed under “film”. Even though this problem may seem trivial or somewhat contrived, it can be an important one, especially if a specific document is to be retrieved.

Another, unrelated issue is raised by the emergence of resource sharing on the Web, more particularly non-textual user-created content. For such documents to be indexed by search engines, they need to be linked to from other Web pages. Even then, the information to store about them is scarce, since they don't consist of words that can be used for indexing. Also, there would need to be many incoming links to a document for it to be ranked highly by the search engine, making the retrieval of a specific resource even more cumbersome. What separates site-specific search tools from Web search engines is their systematic inclusion of the annotations provided by the users. Most of the time, these take the form of tags, metainformation represented as single words and assigned to a document to describe its

¹ One notable exception is the search engine Google. By placing a tilde (“~”) before a keyword, the search results will include Web pages matching similar terms, although mostly not exact synonyms.

content. Unfortunately, their application across different resource sharing platforms is far from uniform, making them difficult for Web search engines to find and identify as metadata.

A possible solution to the two problems discussed here would be to shift the information about the content to the Web page code and to use a more machine-readable syntax. This idea is promoted by the Semantic Web initiative, with the intention of giving information a meaning that can be interpreted without any human intervention. This way, a program could autonomously decide to include some related terms in the query or even answer questions involving multiple resources. However, while the technology necessary to do this exists and has been standardized by the W3C, it is a long shot from being used widely enough to consider harnessing such meta-information with a search engine trying to achieve the goal of finding useful resources. Nevertheless, it remains an option to be reckoned with in the near future.

The potential solution examined in this paper are folksonomies. Folksonomies are a sort of structureless classification resulting from a community of Web users collaboratively annotating resources with tags. Nevertheless, the invaluable meta-information they contain is not recognized as such by Web crawlers, the components of search engine that navigate the Web. Considering folksonomies' high suitability specifically for the classification of user-created documents, they represent a great added value for Web search engines. Also, collaborative tagging's use of a broad vocabulary makes the retrieval of resources easier, since, as a result of a network effect, most of the terms relevant to users will be included in a document's meta-information.

1.2 Objectives

The context described above is first to be extensively discussed in a theoretical part, whereupon the following questions will be attempted to be answered and be at the core of a more practically focused part:

- **What types of folksonomy applications exist and what are their underlying structures?**
- **Which possibilities can be considered to exploit the previously defined different kinds of folksonomies?**
- **With regard to one such possibility, how can it be implemented to work efficiently in a Web crawler?**

The first question will in fact already be partly the subject of the theoretical part. The goal is not to create an exhaustive list of the different representations of folksonomies, neither to enumerate all the imaginable ways these could be harnessed, but to extensively discussed the most relevant and promising instances of each. As for the Web crawler implementation, the objective is simply to explore the feasibility of the discussed content in a practical example.

1.3 Methodology

The theoretical part, which will introduce the subjects, among other related ones, of Web crawlers and folksonomies, will be entirely based on a thorough study of corresponding documents, such as books, scientific papers and a few Web resources. With this theoretical foundation, the intention is then to create a reference model of different types of folksonomies. The applications of collaborative tagging resulting from this simplified illustration are subsequently analyzed argumentatively upon their potential uses for indexing by a Web crawler and different methods to do so will be examined. In order to demonstrate one of the previously discussed methods in practice, a prototype of a Web crawler harnessing folksonomy tags will be implemented as well as evaluated. This prototype will be coded in the programming language Java and the corresponding executable file and source code made available on the enclosed CD-ROM.

Part I:
Introducing the Theoretical Concepts

2 The Web Search Engine

2.1 Introduction

The World Wide Web has become the single largest resource for almost any kind of information. Never has there been a bigger repository of readily available content on any particular subject. It is hard to find an accurate estimate of the size of the Web, but the most recent figures [Alpert & Hajaj 2008] claim there are over one trillion unique URLs for over 110 million registered Web sites [DomainTools 2009] and at least 20 billion pages in the indexed Web [de Kunder 2009]. As opposed to the so-called “deep Web”, the indexable Web (also called “surface Web”) contains the pages that are not dynamically rendered through a form or have access restrictions and can thus be considered for indexing by Web search engines. According to [Bergman 2001], the totality of publicly accessible information on the deep Web was estimated to be 400 to 550 times larger than the surface Web. Despite the inability of search engines to extend their scope to the base of the on-line document iceberg, they are still of considerable importance in helping users find their way in the World Wide Web information chaos.

2.2 A Brief History of Web Search Engines

Search engines represent the most important tool for information retrieval on the Web. Ever since information started being shared worldwide, the task of finding relevant content has been a central part of the users' experience. At the beginning of the Web, a complete list of all Web servers was maintained. But rapidly after the WWW's invention by British computer scientist Tim Berners-Lee, it became impossible to keep this directory up-to-date.

The first search engine, Archie, was actually developed in 1990, prior to the Web's invention, and kept an archive of various FTP host directories by regularly connecting to them and automatically copying the lists of public files. Up to that point, retrieving a file on the Internet was restricted to users who specifically knew where and how to find it, i.e., on which FTP server this file was located. Later on, more and more Web-specific search engines, such as JumpStation, AltaVista or Lycos, appeared and were based on a similar architecture as Archie, in that they kept indexes of regularly visited Web pages.

At first, these indexes were mostly storing on-page data, such as text and formatting, but a later generation of search engines, such as Google (www.google.com), started to include other, more Web-specific data, like anchor-text and link analysis. By taking the structure of the Web into account, it became easier to determine the relevance of pages.

A more current trend goes towards attempting to understand the user's need and adding value to the retrieved information. Search engines such as Rollyo (www.rollyo.com), Swicki (www.eurekster.com) or Clusty (www.clusty.com) try to narrow down the number of results while increasing their quality by clustering the results and using more human input.

Still, current Web search engines have to rely heavily on the type of document formats, which, like HTML, only describes the way a page should be structured and visualized. The future development of the WWW, though, might lead to markup languages which also convey information about the content they represent and would be more easily understandable by computers. This would make it possible for search engines to perform a much wider array of tasks, since basically they wouldn't simply gather data, but also be able to extract knowledge from it. There is more to come on this subject in Chap. 3.

2.3 The Architecture of Web Search Engines

Typically, an on-line search engine features a rather simplistic start page with a blank text field and a search button. Upon entering a set of words, an often extensive list of links is returned, usually ordered by how well the user search criteria are matched. To refine the search, a certain number of operators are usually available to the user. This convenient and easy-to-use tool allows users to very quickly retrieve relevant information and has thus become the dominant information retrieval method. But under the surface, all the processes required to gather and maintain the data to be displayed within a very short response time require much more computational effort than the average user might think. This section will briefly explore what a Web search engine really does and how its different elements and functions operate, and only later have a closer look at the so-called Web crawlers, given their central role in this paper.

2.3.1 A Brief Introduction to Web Crawlers

The results returned by a Web search engine can only be as relevant as the documents stored in its index, hence the importance of gathering a vast amount of pages from the WWW. This process of gathering is commonly referred to as “crawling” and is executed automatically by a computer program called “Web crawler”, “Web spider” or by other terms, which often refers to the way it explores the Web. Starting from a set of seed URLs, the Web crawler fetches a page's source code, extracts the links to other pages and stores them in a list called the frontier. Once all the links have been extracted, the

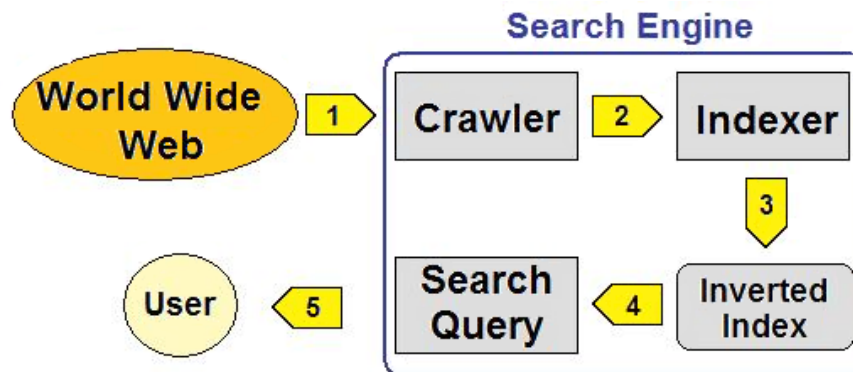


Fig.2.1: A Web search engine architecture

page is stored for further use. Then, the next page in the frontier is fetched and processed in the same manner. This sequence is called the “crawling loop” and corresponds to step one in Fig. 2.1. This goes on until a given number of pages have been visited or, although this only happens on very rare occasions, if the frontier should be empty. The processed pages are finally handed over to a repository, where they will later be accessed by the indexer to be processed as described in Sect. 2.3.2 (step two in Fig. 2.1). Of course, the above description only outlines a very basic and inefficient Web crawler and will subsequently be extended.

2.3.2 Indexing

In order to actually enable the retrieval of relevant documents from the set of stored pages by a keyword search, these are processed through an indexer. The task of this module is to extract

information from Web pages and to structure it, which is a process that is divided into several subroutines. A first step consists in preprocessing the page's source code in such a way that its content can be indexed in a straightforward and consistent way. This step is commonly referred to as “parsing” or “tokenization”, and is faced with the challenge of separating the document's words, displaying them in a uniform representation and also extracting the content from the information specific to the page layout. This is done by removing HTML tags, punctuation marks and so-called “stopwords”, which are frequently occurring yet unimportant terms such as articles and prepositions. Also, all the characters are converted to the same (usually lower) case. Furthermore, words are often reduced to their canonical form in a process that replaces inflected ones with their stem, base or root. For example, the present continuous tense “crawling” would be turned to the infinitive tense “crawl” and the plural “crawlers” to a singular “crawler”. This is called “stemming” and serves to reduce the size of the index and broaden a query by including similar search terms. On the other hand, this can also reduce precision, as some words may be taken away from a specific context.

As the words are being parsed, they are put into their corresponding document's list, called “forward index”, usually along with some information concerning the HTML tag in which they appear, such as font, type of text field and emphasis. Since this data structure would not yet be optimal to quickly retrieve the search terms, the forward index is sorted in such a way that the $(URL, list\ of\ words)$ pairs are transformed into $(word, list\ of\ URLs\ containing\ the\ word)$ pairs to form a more efficient data structure, the “inverted index” (step three in Fig. 2.1).

URL	Words
URL_1	web, crawler, file, document
URL_2	semantic, web, document, search
URL_3	folksonomy, metadata, web, search

a)

Word	URLs
crawler	URL_1
document	URL_1, URL_2
file	URL_1
folksonomy	URL_3
metadata	URL_3
search	URL_2, URL_3
semantic	URL_2
web	URL_1, URL_2, URL_3

b)

Tab.2.1: Simplified examples of a) a forward index and b) an inverted index

Of course, the additional characteristics of the words are also maintained in the inverted index, which would actually make the pairs more like (*word, list of URLs containing the word, additional information*) tuples. With this layout, the terms can be used as access keys to the index, enabling a quick location of matching documents.

2.3.3 The Web Search Query and Link Analysis

Although it would be possible to answer a query by simply consulting the inverted index and returning the list of URLs in the keyword's entry (step four in Fig. 2.1), this would barely yield a satisfactory result, as the possibly huge amount of returned links would be likely to appear in a random order and force the user to browse through all of them to find relevant information. This problem is solved by presenting the user an ordered list with the documents most likely to satisfy his need for information first (step five in Fig. 2.1). Of course, in a query with more than two search terms, computing the intersection of the inverted indices of each term would already significantly narrow down the set of returned URLs and allow for a crude way to rank the documents by listing the ones with the most matching words on top. Still, nothing would guarantee the quality of top-ranking results, since a page could feature all the keywords but otherwise be devoid of valuable or reliable content. Some algorithms go on to include the frequency or the location of the search term in the document as parameters, weighing terms appearing more often or in more prominent fields of the page, such as in titles or headers, more heavily. Another frequently used approach is link analysis, which, although it also cannot conclusively determine the subjective relevance of a document to the user, can provide an objective measure of its importance within the Web. More precisely, link analysis algorithms rely on the cross-referencing structure of the Web to assess the authority of documents by computing a value in dependence of the number of links pointing to it. The assumption is made that a link from a page a to a page b is equal to an assertion of authority of page b by the author of page a and thus that a large quantity of links to a document reflects its quality. The better a page's score, the higher is it to be ranked on the result list. Additionally, links from pages with a high score may be given higher weight. Two well-known link analysis algorithms that will briefly be discussed are HITS and PageRank.

The HITS (Hyperlink-Induced Topic Search) algorithm is designed to be run at query-time on an initially compiled set of result pages, not on the entire Web. Only the link structure of the relevant part of the Web graph is analyzed. In this subset two weights are computed for each page, a “hub score” and

an “authority score”. In the HITS terminology, a “hub” is a page with many outgoing links, whereas an “authority” designates a page with a lot of incoming links. Hubs and authorities define each other's score in a mutual recursion, as a hub is given a good score if it points to many highly ranked authorities and inversely an authority is given a value depending on the number of high scoring hubs pointing to it. HITS has the advantage that pages are evaluated in the context of a query topic by initially performing a content-based sorting and only then determining a link-based rating, which allows it to provide a higher level of relevance. A weakness of the HITS algorithm is that it can easily be influenced by adding many outgoing links to good authorities to increase the hub score of a page. Topic drift is another major problem that concerns HITS. In the HITS paper [Kleinberg 1999], the root set R containing 200 pages matching the search term is expanded before computing hub and authority scores by including any page pointed to by a page in R and each page in the root set brings in up to 50 pages linking to it. This expanded set is called the “base set” and serves to include a wider array of good hubs and authorities, but has the drawback that many pages that are unrelated to the search topic may be collected, because links don't necessarily connect pages covering similar subjects. Also, computing the ranking algorithm at query time is likely to noticeably slow down the search engine performance.

The link-analysis algorithm used by the Web search engine Google is PageRank. Like HITS, it also analyzes the link structure of the Web, attributing a value, simply referred to as the PageRank, to a page based on its position in the Web graph, but unlike the previously discussed algorithm, this value is computed statically, before a search query is done by an user. To determine the PageRank value of a page, only inbound links are taken into account.

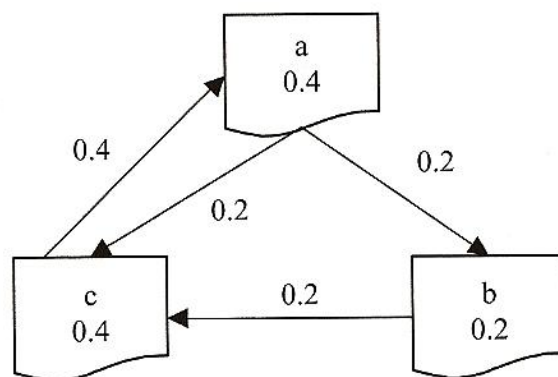


Fig.2.2: A network of documents with their PageRank score [Markov & Larose 2007]

This value can be considered as an expression of the probability that, starting from a random page and by clicking random links, a user will visit the page. A parameter, the “damping factor”, is used to include in the equation the possibility that the random surfer will at some point stop following links. Initially, in a set W of n pages, each page would be attributed a value of $1/n$. Considering the example of Fig. 2.2, this value would be $1/3$, or 0.33. The PageRank is computed not only from the number of incoming links, but also depends on the value of the pages that contain those. A PageRank is divided by the number of outgoing links from a page, which is the value that is passed on to the linked pages. To illustrate this, let us go back to the network of Fig. 2.2; Taking document a as an example, its own PageRank value equals the sum of the values of the pages linking to it, which is only the incoming link from c . This PageRank of 0.4 is then equally propagated among the two outgoing links, conferring page b and page c a value of 0.2.

As in HITS, a large number of incoming links from highly rated pages in turn attributes a page a high value, but since outgoing links are disregarded in the calculation of a page's value, PageRank is more difficult to influence and less spam finds its way into the top-ranking search query results. Also, the fact that the ranking value is computed off-line considerably improves the search engine's performance. The downside of having a query-independent measure is that no distinction is made between pages that are authoritative in general and pages that are authoritative on the query topic. Another question is how the PageRank algorithm deals with the dynamic nature of the Web and with changed or deleted contents.

Other major search engines use their own algorithms, which are usually not revealed, as they are trade secrets and also because their publication would allow spammers to exploit them in order to increase traffic to their site. There are also differences in the way results are displayed and in the features offered to help the user with his search. Many search engines include advertisement related to the query topic or generate revenue from advertisers willing to pay for a higher ranking.

2.3.4 Query Categorization

Web search queries are classified into three categories; informational, navigational and transactional. An informational query is one with the purpose of finding information on a subject present in one or more documents. This type of query is often formulated very broadly and there may be a large number

of relevant results, depending on the specification of the query topic.

The second type is the navigational query. Here, the intent is much less ambiguous, as the user is looking for a specific page he visited previously or assumes exists, such as a company's homepage. There is usually only one relevant page in the set of results returned by the search engine.

In transactional queries, Web sites where some activity can be performed are looked for. This differs from an informational query in that the user isn't mainly interested in some pages' content, but in the services they offer, such as shopping, downloading a file or querying a database. One or more pages may be relevant in a transactional query and often the user only looks for a Web site providing a given service and searches the wanted item locally.

What the navigational and the transactional query types imply for a search engine, is that not only the content of a page should be indexed, but also some contextual information, such as anchor texts and URLs, which also reveal a great deal about the nature of a document. According to [Broder 2002], most Web search queries belong to the informational type, followed by transactional and navigational queries.

2.4 The Web Crawler

As seen previously, Web crawlers are automatic programs that exploit the link structure of the WWW to hop from page to page by collecting all the references to other pages and adding them to a to-do list called “frontier”. The visited pages are then passed on to the indexer. Because the Web is a dynamic entity with documents constantly added, modified or removed, and due to the huge number of pages on it, there is a continuous need for crawlers to explore the Web and to keep their client up-to-date. As a matter of fact, although most of the time they serve a search engine, Web crawlers can be used in many different applications, such as monitoring a set of Web sites or collecting e-mail addresses for spamming.

2.4.1 Fetching and Parsing

To fetch a page, a crawler takes the role of an HTTP client to send out requests to Web servers. The first step, as for any Web client, is address resolution, which means converting the URL into an IP

address, and is done by sending a request to a DNS (Domain Name System) server. To increase efficiency, addresses already visited can be stored in a local cache, as DNS lookups can have a considerable latency. Another similar issue is the time wasted between a request for a page and loading it. To avoid waiting for too long, the downloading of a page can be restricted to its first 10-100 KB and timeouts defined to avoid too much time being spent on a slow server. Furthermore, multiple pages can be fetched simultaneously to prevent the crawler from being idle (cf. Sect. 2.4.6). Fetching pages is an error-prone process and requires the crawler to be able to do robust exception handling. After fetching a page, its content needs to be parsed in order to extract the URLs to be added to the frontier. This step may also include extracting information for indexing, although in this description, it will be implicitly assumed that the Web crawler and the indexer are two distinct programs. Here again, a robust design is crucial, as HTML source code often contains wrong or misspelled syntax, as browser are very forgiving of such mistakes and crawlers also need to be. Depending on the crawler's task, different levels of sophistication may be required and if only links must be extracted, a bad syntax can be ignored. Otherwise, the rather complex process of tidying up the HTML code has to be performed, where incomplete, missing or misplaced tags are inserted or reordered. Fig. 2.3 illustrates the processes within a crawling loop.

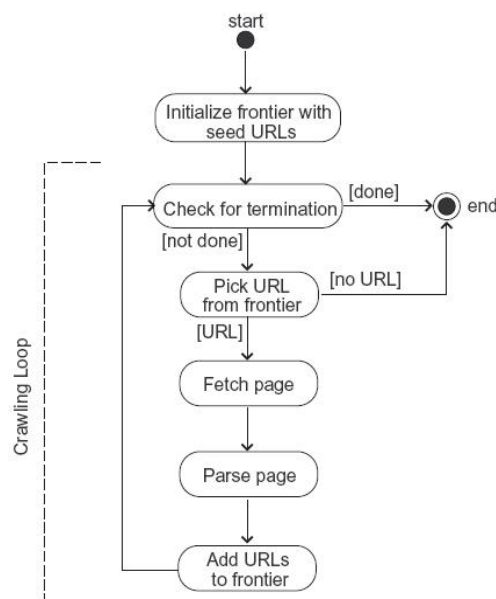


Fig.2.3: Flow of a basic sequential crawler [Pant et al. 2004]

2.4.2 The Frontier

The frontier is the Web crawler's main data structure. The initial URLs in the list are provided by the user or a client program. Rapidly, a large number of URLs will be added to the frontier. Despite the fact that memory capacity isn't a major issue anymore, the frontier is limited in size, as a too large list would be likely to decrease performance. Hence, a policy must be put in place to decide how to proceed once the frontier fills up. One possibility is to implement the frontier as a first-in-first-out (FIFO) queue, where no page is ever discarded, except of course if it has been visited. The next URL to visit is the oldest one in the frontier, and if it is already full, only one new page can be added to the queue. An illustration of a simple FIFO frontier holding up to six URLs is given in Fig. 2.4: The first step consists in adding the extracted links to the end of the queue. The next URL to crawl is removed from the start of the queue (step 2) while the following entries are advanced in the queue. In such a design, priority is given to exploring the Web graph in its breadth dimension. When a breadth-first crawler is used, the Web graph can be visualized as a tree, with the seed page at the root and the crown of the tree growing larger and larger as each pages links to several others, until, at some point, the graph reaches its maximum breadth, as the frontier is full.

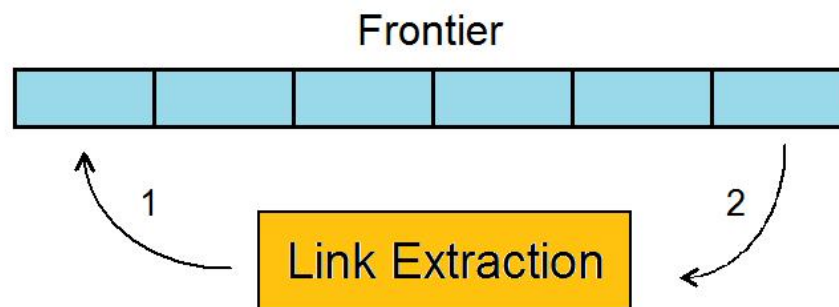


Fig.2.4: A frontier implemented as a FIFO queue

Such a representation does in reality not reflect the nature of the Web graph, since typically a page is pointed to by more than one link, but in a single crawl a page is visited only once. To do so, a Web crawler needs to make sure that no URL already present in the frontier is added to it. Looking up each frontier entry each time a new URL is extracted from a page is a rather time-consuming operation. To

avoid this, a solution for a faster lookup is to hash the URLs and to keep them in a separate table that is kept synchronized with the frontier. A crawl history, which will be discussed shortly, also needs to be maintained. Having the frontier implemented as a FIFO queue may mislead to the conclusion, that the pages are visited in random order. But since popular pages have the most incoming links, they are likely to be visited early in the crawl. The goal of a breadth-first search is to exhaustively traverse the Web graph, or at least the linked nodes, as it may be the case for a general-purpose search engine.

But, as mentioned before, a crawler can also be applied in contexts other than within a Web search engine. Usually, the goal is then to explore only the portion of the Web graph that is relevant. Since the focus is no longer to visit as many pages as possible, but to download only a select few, the frontier is evidently also implemented differently. A strategy must be put in place to decide which URLs to add to the frontier and in what order they are to be visited. Such a design in which the frontier is specified as a priority queue is known as a preferential (or best-first) crawler. The main design concern for a preferential crawler is how pages can be prioritized and the relevance of its content determined before even visiting it. This value can be assigned on the basis of some heuristic and can vary widely depending on the crawling algorithm applied. When a best-first crawler's frontier is full, a URL can only be added if it is not already present in the frontier and if it is estimated to be of a high enough relevance to be ranked amongst the top *max* URLs, with *max* being the size of the queue. Sect. 2.4.5 goes into more details on ways to prioritize URLs.

2.4.3 Additional Crawler Functions

To ensure no URL is visited a second time after it has been removed from the frontier, a history of the crawled pages must be kept and updated each time a page is fetched. The same issues as in the frontier apply regarding the optimization of lookups, as a hashing of the URLs might also prove useful in this case. The crawl history may also be used to analyze the path of the crawler through the Web and to evaluate the crawler's performance. Another mechanism to avoid fetching the same page twice is canonicalization, the process of converting a URL into a canonical form. This is required because different URLs could point to the same page for various reasons. For one, characters can be displayed in different ways, such as in lower or upper case and as encoded characters (“%7e” instead of its ASCII counterpart “~”). Another reason is irrelevant additional information in the URL, like the anchor part (characters following a pound sign in the URL), file names that lead to default pages (often

“*index.html*”), parent directories of files (“..” has to be removed) and default port numbers indicated in the URL. How specifically a crawler handles these different cases doesn't matter as much as being consistent in doing so. For some cases, heuristic rules must be applied, where there is a trade-off between making wrong assumptions and discarding possibly unvisited pages and missing URLs pointing to the same page because of insufficient transformation. On some occasions though, canonicalization is not enough to avoid returning to a previously visited page. On certain sites, links are created dynamically depending on the path a user followed or his previous actions on the site. For example, an on-line store may suggest the user a product based on pages about similar articles he just visited. This dynamically generated link on a page x contains information telling the next page fetched y where the user came from, making it impossible for the crawler to distinguish the URL from another one pointing to the same page. If page y in turn also contains a dynamic link to page x , the Web crawler is trapped in an endless loop. This is called a “spider trap” and leads to an unwanted situation for both the crawler and the server, who see their respective bandwidths wasted away, and may end with a filled up database on the crawler side or with an inability of the server to answer any further requests. A solution for this problem is to limit the acceptable length of a URL, since as the crawler keeps visiting the same pages, their URL grows larger and larger.

2.4.4 Crawler Etiquette

An even simpler approach than determining a maximum URL length is to restrain the number of pages a crawler can request from the same domain. Not only does this decrease the impact of a spider trap, but also frees up network resources and server capacity, as crawlers can send out consecutive page requests much quicker than human users and slow down the servers for them. This is also why not only the number of requests may be limited, but the time interval between two requests from a server may as well be defined in such a way that server overload is prevented. Crawler etiquette, as such measures are often called, also requires a crawler to disclose information about itself in a HTTP header field and to abide by the “Robot Exclusion Protocol”, a standard used by Web server administrators to communicate to crawlers which files they are allowed to access. An optional file named “*robots.txt*” is kept under the Web server's root directory and contains the access policy instructions in a specific format. A *User-agent* field specifies which particular crawler an instruction is directed to (an asterisk, as in the second *User-agent* entry in Tab. 2.2, matches all the crawlers that do not match other *User-*

agent values) and a number of following *Disallow* entries determine which pages are prohibited from being fetched. A slash prevents all the pages from being fetched, as it is the case for the unwanted crawler in Tab. 2.2.

```
User-agent: ImpoliteCrawler
Disallow: /

User-agent: *
Disallow /images
Crawl-delay: 5
```

Tab.2.2: A robots.txt example

To comply with this standard, a crawler has to check for a *robots.txt* file before requesting for a page. If no such file is present, it may be assumed that all pages are open for access. However, if there is a *robots.txt* file and the crawler matches a *User-agent* field, none of the pages mentioned in the following *Disallow* entries may be fetched. Other, non-standard fields include *Crawl-delay*, which sets the number of seconds to wait between successive requests and would be of five seconds in the preceding illustration and *Sitemap*, whose value is a URL of an XML document that lists the site's pages available for crawling along with some metadata about each of those pages. Compliance with the Robot Exclusion Protocol is voluntary, the standard relies on the cooperation of the crawler and cannot be enforced, since a page accessible for a regular user is just as accessible to a crawler. However, crawlers that repeatedly display an impolite behavior are likely to be blocked by servers based on their IP address.

2.4.5 Crawling Algorithms

As seen in Sect. 2.4.2, the most important element in determining a crawler's path through the Web graph is the frontier. Its implementation as a FIFO queue results in a breadth-first crawler and the variations in its crawling algorithm are restricted to the setting of parameters like the size of the queue, the maximum number of links to extract from a single page or connection timeouts. In a best-first search, however, not only parameter settings, but also the heuristics used to score unvisited URLs allow for a wide array of different crawler designs.

One such design is a focused crawler, which attempts to classify crawled pages into categories. Starting with a topic taxonomy and a set of seed URLs, the crawler is biased towards pages belonging to a certain category. One approach is for the crawler to predict the relevance of a page based on the link's anchor text or on other content from the page holding the link. In a method described by [Chakrabarti et al. 1999], the relevance may also be determined after downloading a page by using a classifier that applies some Bayesian probability function, that is, a probability prediction based on some prior condition. Focused crawlers make use of the fact that pages on a given topic are often found in a single portion of the Web graph, that is, they all link to each other. This “topical locality” phenomenon can be harnessed to point the crawler to a set of good pages on a topic and have it start the navigation from there. Another similar type of crawlers, “context-focused crawlers”, also uses Bayesian classifiers, but additionally has them make an estimation on the distance between a visited page and relevant pages. An otherwise low-scoring, presumably irrelevant page, might then be kept in the frontier, because of the assumption, that it is located in a promising neighborhood of the graph.

Generally, when a crawler is able to learn from an input, such as crawled pages or user-defined topics, and adapt its behavior accordingly, it is said to be “adaptive”. The ability to exploit information other than merely content and to predict the benefit of visiting a page with a specific focus is what makes preferential crawlers different and so much more complicated than traditional crawlers.

2.4.6 Web Crawler Challenges

The crawling mechanisms described so far have all been represented as a series of sequential procedures. In reality however, such a design would be far too inefficient to explore a large part of the Web within a reasonable amount of time. The resource used by a sequential crawler, network, CPU, and disk, represent bottlenecks when consumed and lay idle the rest of the time. A considerably more efficient way to design a crawler is to run concurrent processes or threads. Multiprocessing and multithreading each have their advantages; the former may be easier to implement, whereas the latter is possibly less costly, as creating and destructing threads doesn't require trapping to the operating system and idle or terminated threads always allow for other crawling threads to run, which may not be the case with multiprocessing, because an outside process can be chosen to be run next by the CPU scheduler. In a concurrent crawler, each thread or process works as an independent crawler executing crawling loops just like a sequential crawler. The main difference is that to prevent fetching a page

already visited in another loop, the frontier must be shared. To avoid race conditions, a frontier manager is responsible for locking and unlocking access to the frontier, as well as to other shared data structures, such as the crawl history. If the frontier runs empty, loops that want to fetch another page are put to sleep, as there might still be other loops processing pages and subsequently adding new URLs to the frontier. The number of sleeping processes or threads must be kept track of and the crawler stopped if all of them have reached a dead-end. Especially with concurrent loops with different starting points, this event seems very unlikely. Multiprocessing or multithreading can considerably speed up a crawler, as resources are made much better use of. To scale up to the kind of performance required for a commercial search engine, additional measures need to be implemented. For one, with the use of asynchronous sockets, network calls can be made non-blocking, that is, a process or thread can hold multiple connections simultaneously. Evidently, such a design obliterates the necessity of having multiple processes and threads as well as locking frontier access, as a single instance can handle hundreds of connections without ever blocking to wait for a response.

Other performance-increasing methods exist, including many approaches to minimize the overhead of connections. Along with the size of the Web, its dynamic nature presents a major challenge to crawlers. One question raised by the constantly evolving and newly emerging Web features and services is how the crawler adapts and responds to such changes. To perform well, it needs to be implemented in a very robust way and be updated regularly. Also, maintaining an index of such a vast and dynamic collection of documents as the Web is and keeping it up-to-date is a difficult task. To keep the index “fresh”, i.e., based on current copies of the indexed Web pages, as they would appear if they were accessed on-line, the pages need to be revisited after a certain amount of time. Different re-visit policies exist to determine the frequency of page crawls. Two simple policies are discussed and compared in [Cho & Garcia-Molina 2003]. In the first one, it is assumed that all pages are changed with an uniform frequency and thus all need to be visited with the same frequency. The premise of the second policy is that pages change at different rates and that these rates are known. In this case, the pages must be visited with proportional frequencies. The study surprisingly concludes that, in terms of average page freshness, a uniform re-visit policy performs better than a proportional one, because with the latter policy, too much time will be spent on visiting very frequently changing pages that yield less benefit, because most of the time the last indexed copy will not be fresh anyway. However, not having fresh copies of pages doesn't necessarily mean that a search engine will return useless links, as pages are often only changed by the addition of new content, without removal of the old one. The new content

will then simply not be indexed.

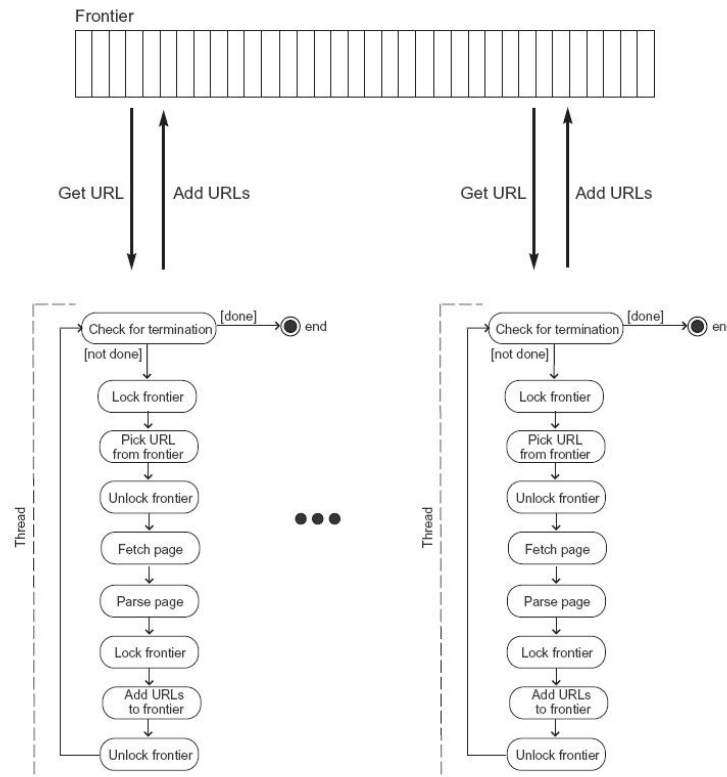


Fig.2.5: A multithreaded crawler model [Pant et al. 2004]

Underlying the concept of crawling the Web is the assumption that all the nodes on the Web graph are somehow connected to each other and thus indexable. As briefly mentioned in Sect. 2.1, this does not hold for a very large part of the WWW. The deep Web, which also contains pages that are dynamically generated, for example as a result of a query, or pages with a restrictive access policy, is per definition not indexable by a regular Web crawler. Still, more and more search engines have found ways to integrate previously inaccessible content into their index.

The future development of Web crawlers is closely related to the future of the World Wide Web altogether, as it will always be forced to evolve in lockstep with the changes undergone by the Web standards and adapt to make the best out of emerging technologies. A very promising current and future evolution is the Semantic Web, which provides new ways for crawlers to retrieve and for search engines in general to identify content. Folksonomies may also turn out to be an invaluable help for the indexing of user-generated documents and are the main subject of the following chapter.

3 The Semantic Web and Folksonomies

3.1 Introduction

As mentioned earlier, the Web keeps moving away from its original static design and towards a more dynamic and user-centered paradigm that fosters resources which aren't as easily accessible through a keyword-based search. Along with the increased interactivity comes the need to classify newly generated user content in a consistent way. Folksonomies are a widespread and popular way of letting the user communities themselves collectively categorize documents, but also represents a considerable challenge to Web crawlers willing to harness this valuable information on some content but unable to identify it as such. Folksonomies are the main subject of this chapter, although the term “metadata” and its significance will first be introduced followed by a discussion of the related topic of ontologies in the context of the Semantic Web.

3.2 Metadata

Metadata, or metainformation, is data that is not part of the content of a document, but rather data about the content and the document itself. Its main use is providing a possibility to bridge the gap between heterogeneous types or formats of documents through a homogeneous and representation-independent abstraction. A good example for this are barcodes: they allow to label a wide range of different items with a uniform representation of data. Both the Semantic Web and folksonomies aim at putting documents into into a context and provide models of how metadata can be applied.

3.2.1 A Classification of Metadata

If the above introductory description is rather vague, it is because metadata can be found in many different forms. [Kashyap & Sheth 1997] propose the following classification:

- **Content-independent metadata:** Metadata that is completely independent of its document's

content and contains contextual information, such as the author or the date of creation of a document.

- **Content-dependent metadata:** This type of metadata does not relate directly to the content, but rather to attributes derived from it. The size or the language of a document are examples for content-dependent metadata.
- **Content-based metadata:** An index, as used by a Web search engine, is an example of metadata that is directly based on the content and which adds information or structure to it.
- **Content-descriptive metadata:** This type of metadata is of particular relevance in this paper, as folksonomies belong to it. Content-descriptive metadata provides, in a more or less well-defined form, a description of the content, which may range from sets of keyword to the attribution of a document into a thematic category.

Each type of metadata mentioned here covers aspects that may be important in different applications. In the example of a Web search query, a user might want to look for attributes concerning any of these types, although typically content-based information is investigated. Furthermore, identifying content-descriptive metadata represents a major challenge for Web crawlers and various approaches to do so will be considered in Chap. 4.

3.2.2 Benefits and Limitations of Metadata

For many information repositories, the heterogeneity of the documents contained in it represents a challenge to organize. Metadata allows to look beyond representational differences and focus on a uniform method of displaying information. It provides a possibility of comparing and ordering arbitrary documents along some common set of attributes, such as topic, author or age. Aside from the obvious purposes of structuring and maintaining metadata is able to support, it is also crucial in finding information sources in large data sets, such as the WWW.

Although there actually is a syntactic standard provided in HTML to associate metainformation with a document, it is not given much or any weight at all anymore when indexed and ranked by a Web search engine. As seen in the previous chapter, search engines prefer to generate their own metadata on Web pages, typically some form of content-based information. This is due to the fact that when these so-

called “meta-tags” were still taken into account in the mid-1990s, search engine rankings could be easily influenced by Web masters interested in generating more traffic to their site. The keyword list in the meta-tag is very unreliable, as there exists no restriction on keyword repetition and spamming and is likely to be used with the intent to mislead search engines. However, other meta-tag attributes, like the page title, author or content description, are still widely used, but only on a voluntary basis. Other metadata standards exist, such as Dublin Core, which specifies a list of 15 metadata elements for a wide range of resources.

The problem with managing and searching mechanisms that rely completely on metadata is that the entire set of documents needs to have corresponding metadata, otherwise possibly important resources might be left out. Also, this metadata has to be maintained in a consistent way across all the documents, that is, the metadata must appropriately and objectively reflect a resource's content to avoid having it categorized incorrectly. This in turn means that the interpretation of the content needs to be done consistently. Another requirement is for metadata to be accessible easily and uniformly. It can be stored either right alongside the corresponding document, like the meta-tags in the HTML code, or externally from the information source in a central storage unit, as it may be the case for a data repository. The latter method allows a more efficient search and less redundancy, but is typically harder to manipulate, especially linking to the document it is affected to.

3.3 The Semantic Web and Ontologies

It may seem unusual that in this paper, the more recent Semantic Web initiative, which some consider a Web 3.0 concept, is discussed first and given somewhat less attention than the Web 2.0 approach of using folksonomies. Although both of these methodologies have distinct applications, it might be argued that they cover broadly the same issue of giving context to Web documents and can theoretically both be harnessed by a Web crawler. Also, the Semantic Web is in fact designed specifically to provide machine-readable context information and has the potential of revolutionizing Web crawling. Nonetheless, it was decided to focus on the methodology already widely accepted and in use, that is folksonomies. Despite the promising future of the Semantic Web movement, it still has a long way to go to become the dominant model of the WWW.

3.3.1 The Purpose of the Semantic Web

The results of a Web search query when applying the indexing mechanisms described in Chap. 2 are likely to be of good relevance, as it is easily possible for some algorithms to determine in an ad-hoc fashion how well a document matches some search terms based on full-text indices and analysis of the Web graph. However, these mechanisms could be made much simpler if there was a possibility to determine the context of a page's information in a well-defined, uniform way. More precisely, a consistent definition of the meaning of information would have to be provided and in such a way that it could be understood and processed automatically by a machine (such as a crawler). This is exactly the goal of the Semantic Web movement. The term “Semantic Web” is derived from “semantics”, the study of meaning, and is precisely what this new development is about; giving a meaning to information, that is, a meaning that can be perceived by a computer program. Thus, the Semantic Web doesn't propose a change in the way pages are displayed by a browser, but a change in the way information is structured and formatted, a change that is hidden away in the markup language (see Sect. 3.3.2 on Semantic Web standards).

Basically, the purpose of the Semantic Web is to enable machines to recognize and interpret contexts the same way a human does, that is to make sense of information and, most importantly, to connect and combine it. Then only is it possible to harness the full power of the Web, when information can be understood, compared and cross-referenced automatically. Let us consider, for example, a user planning a trip to a foreign country on-line. Most likely, booking a flight, making a hotel reservation and finding information on the destination is going to take some time, as many different sites need to be visited and compared. Also, all the data found on pages differing in language, style or content has to be integrated and cross-referenced by the user. If there were a standard to put data into context, an application could conceivably automatically interpret and gather all the wanted information and display them in a personalized way, even document formats that aren't textual, such as images or videos. The machines' ability to tell precisely what kind of information they are dealing with is crucial, as it allows them to specifically find what it is looking for in a set of Web pages and exploit it themselves. Search engines could for example provide answers to queries spanning multiple information sources and directly integrate them.

3.3.2 Semantic Web Standards

To enable the concept of the Semantic Web and uniform information representation, standards are needed. It was already mentioned before that providing context to a document requires a different approach to the markup convention. Indeed, the currently predominant standard, HTML, provides no solution to allow a program to interpret the significance of a piece of information. As its Semantic Web counterpart, XML (Extensible Markup Language) is used. This language provides a format to define data structures and allows the user to define his own tags, but is only syntactical. To give information its meaning using XML syntax, the “Resource Description Framework” (RDF) has been proposed. RDF is a generic format to represent meta-information and one of its key features is its use of “Uniform Resource Identifiers” (URIs) to reference resources. A URI is a string of characters that uniquely identifies any resource (which may be an entire or just part of a Web page) on the Internet and facilitates the integration of nonlocal items. The RDF data model is called a “triple”, which consists of three URIs; a subject, a predicate and an object. These are respectively a resource, an aspect of this resource and the value corresponding to this aspect, and have a labeled directed graph as their underlying structure. To exemplify an RDF statement and its syntax, let's consider the sentence “Thomas Vander Wal is the author of *http://vanderwal.net/folksonomy.html*”.

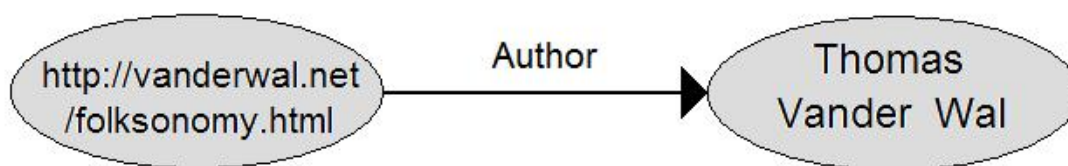


Fig.3.1: An example RDF graph

The sentencing might be misleading, as in this example, the URL is the subject and the author the object, which may in fact be a name (represented as a character string), an e-mail address or a structured entity that is itself part of multiple subject-predicate-object relationships. Tab. 3.1 displays the RDF/XML representation of the same example sentence.

```

<rdf:RDF>
  <rdf:Description about="http://vanderwal.net/folksonomy.html">
    <s:Author>Thomas Vander Wal</s:Author>
  </rdf:Description>
</rdf:RDF>

```

Tab.3.1: An example RDF/XML statement

An *RDF* element marks the boundaries of the XML document whereas the *Description* element identifies the resource being described. The usual RDF namespace prefix is *rdf:* and other arbitrary prefixes, *s:* in the above example, can be chosen and defined in an XML namespace declaration. Of course, many other elements exist, such as *type*, which captures the class-instance relationship, but will not be addressed in further detail. Serialization formats other than RDF/XML also exist.

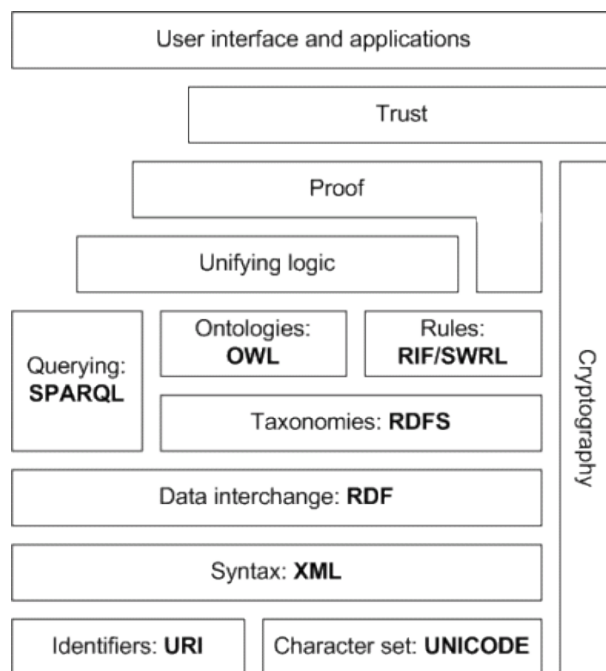


Fig.3.2: The Semantic Web architecture [Obitko 2007]

To be able to include user-defined taxonomies of classes and properties, RDF vocabulary needs to be extended with RDF Schema (RDFS). RDFS allows to create hierarchies of classes and properties in order to structure RDF resources and is in fact the lowest layer in the Semantic Web architecture (cf. Fig. 3.2) that represents machine-readable semantic information. The Web Ontology Language (OWL),

which was standardized by the W3C in 2004, in turn, extends the somewhat limited expressiveness of RDFS. Three sublanguages of OWL are defined and basically provide ways to represent, additionally to RDFS, certain constraints on sets of “individuals”, as data elements described in OWL are called. A more detailed discussion of how OWL works and what ontologies exactly are can be found in Sect. 3.3.3. Finally, the query language for Semantic Web resources, such as RDF data, RDFS or OWL ontologies, is SPARQL (SPARQL Protocol and RDF Query Language).

The basic concepts underlying the Semantic Web architecture are shown in Fig. 3.2. The layers above those discussed are still a subject of ongoing research. The goal of these are to provide ways for computers to make logical deductions and be able to prove them. Another objective is to be able to trace back content to verifiable sources using cryptography. With these layers effectively working, user applications can then be built on top.

3.3.3 Ontologies in the Semantic Web

Ontologies are at the very core of the Semantic Web, providing the framework to classify and assert meaning to information. The term “ontology” is defined as the study of “the nature of existence or being as such” [Dictionary.com 2009] and in information science refers to a shared model of some real world aspects. According to the most frequently quoted description, “an ontology is an explicit specification of a conceptualization” [Gruber 1996], with a conceptualization being all the objects and their properties and relationships contained in a domain of knowledge. An ontology thus provides the vocabulary to model these. To put it simply, an ontology is similar to a language, as it defines the terms, the relationships and the rules to use it. For example, one could imagine an ontology that models the human anatomy. Object instances, the most basic component of an ontology and often referred to as “individuals”, can be defined along with some properties and attributed to a class. In the anatomy example, the femur bone could be classified in the *bones* class with attributes such as *name* or *length*. The value of an attribute can be any type of individual or some simple character string. The option to link individuals to each other is the main feature that separates ontologies from taxonomies. An ontology also specifies which relations an individual of a certain class can have with other individuals. The *femur* individual might for example have a *isLinkedTo* relation with a *knee* individual belonging to a *joint* class. Such relations may also be subject to some rules and restrictions, for instance that an individual of the *bone* class cannot have more than two *isLinkedTo* relations. Rules can furthermore be

applied to classes or relations and classes be assigned to parent-child relationships with inheritance of attributes.

Ontologies are encoded into a formal syntax using an ontology language. The Web Ontology Language (OWL) is the chosen standard to assert meaning to information in the Semantic Web. OWL comes in three different levels of expressiveness, OWL Lite, OWL DL and OWL Full. While OWL Lite only provides simple features for classification and constraint definition, OWL DL and OWL Full are progressive extensions of it. OWL DL, which owes its name to its membership with the Description Logic research field, supports maximum expressiveness but has, unlike OWL Full, semantic constraints to make sure every kind of computation can be completed. The syntax normally used by OWL and endorsed by the W3C is based on RDF/XML.

A typical initial component of an ontology is a set of namespace declarations, which define the specific vocabulary to be used and identify URLs associated with different prefixes, for example, the declaration `xmlns:anatomy = "http://www.anatomy-example.com/anatomy#"` says where the namespace of the ontology with the prefix `anatomy:` can be found. Likewise, the namespace of OWL is located at `http://www.w3.org/2002/07/owl#`. An ontology header, with information such as comments, version control or inclusion of other ontologies, is then provided within an `owl:Ontology` tag. The most central concepts of ontologies, classes, may contain individuals and other subclasses. Tab. 3.2 shows a possible class declaration.

```
<owl:Class rdf:ID="Organ">  
  <rdfs:subClassOf rdf:resource="#Tissue"/>  
</owl:Class>
```

Tab.3.2: An example class declaration with OWL

In the example of Tab. 3.2, the class *Organ* is defined to be a subclass of *Tissue*. This parent-child relationship is actually defined in RDF Schema, which provides some basic features to define ontologies. More refined class constructors are available in OWL, such as the Boolean operators *and*, *or*, and *not*, which are called, respectively, *intersectionOf*, *unionOf* and *complementOf* in OWL. Individuals are instances of a class, and there may be any number of them. An individual is introduced by simply stating its membership to a class in a tag prefixed by `owl:Thing`. Apart from taxonomies,

OWL of course also provides the tools to define properties, which can be organized in hierarchies, too. Properties can either be a relation between two individuals (object property) or a relation between an individual and some datatype (datatype property). Tab. 3.3 illustrates the former, declaring a bone to be linked to a joint.

```
<owl:ObjectProperty rdf:ID="isLinkedTo">
  <rdfs:domain rdf:resource="#Bone"/>
  <rdfs:range rdf:resource="#Joint"/>
</owl:ObjectProperty>
```

Tab.3.3: An example property definition with OWL

The *domain* field specifies what class a property is defined for, while the value of the property is defined in the *range* field. It is allowed to include multiple *domains* (or *ranges*), meaning that the property is asserted to the intersection of the mentioned classes. A subproperty is a specification of a property, which means the classes a subproperty refers to are subclasses of the ones in the parent property. OWL properties included several features, such as cardinality restrictions, which is exemplified in Tab. 3.4.

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#isLinkedTo"/>
  <owl:maxCardinality rdf:datatype=
    "&xsd;nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>
```

Tab.3.4: An example cardinality restriction on a property with OWL

The above code would be embedded in a class definition body within a *subClassOf* statement, expressing that there is an anonymous parent class of which the individuals are subject to this restriction. The restriction states that the subject of the property, an individual of the *Bone* class, may only hold two (expressed as an XML Schema datatype) relations of the type *isLinkedTo*.

In order to be used on a broad scale, ontologies need to be able to extended and merged. To do so, classes and properties from different ontologies have to be integrated. One important OWL tool that

supports this is *equivalentClass*, which allows to tie two classes together, with or without excluding some instances. To create a new class containing individuals from separate ontologies, one or multiple common property must be defined. Let's assume two ontologies covering different aspects of the human anatomy are to be merge. Tab. 3.5 shows the example of a new class for all the individuals with the value of *isVital* set to *false*.

```
<owl:Class rdf:ID="Non-vital anatomical features">
  <owl:equivalentClass>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isVital"/>
      <owl:someValuesFrom rdf:datatype="&xsd:boolean">
        false</owl:someValuesFrom>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Tab.3.5: An example *equivalentClass* declaration with OWL

The property *equivalentClass* expresses that two classes have the same instances, that is, in the above example, the class initially defined and the anonymous class specified in the *Restriction* boundary. The prefix *someValuesFrom* indicates an existential quantification (“there is at least one individual with the value *false* for the property *isVital*”), whereas for an universal quantification, the prefix *allValuesFrom* would be applied (“for all the individuals with the property *isVital* defined, the corresponding value would be *false*”).

OWL makes an open world assumption. For one, this means that missing information is regarded as unknown and not as false, as an ontology has no minimum content requirements. Also, the description of a resource can be arbitrarily extended. Both of these aspects give OWL ontologies great, unconstrained flexibility while still being confined to a consistent formal syntax.

3.3.4 The Future of the Semantic Web

Having now an understanding of the possibilities of a Semantic Web, it is easy to see its advantages over the current model. However, there a still a few challenges it has to overcome. While OWL has

already been adopted by numerous scientific communities to create ontologies, the Semantic Web standards agreed upon by the W3C are still very recent and will require some time to gain general acceptance. The success of the initiative also depends on the availability of tools to address the enhanced complexity of creating Web content. A similar concern is the cost of ontology creation and maintenance. Furthermore, the question remains, whether biased metadata annotation, be it purposely so or not, wouldn't hinder the feasibility of a Semantic Web that relies on trusted information. The answer to this could by a large part be given by future standards and extensions of OWL to also cover the upper layers in the Semantic Web architecture (Fig. 3.2).

A possible step towards a Semantic Web is the use of a folksonomy and more notably machine tags (cf. Sect. 4.3.2).

3.4 Folksonomies

While ontologies work perfectly fine for scientific classifications, where the metadata needs to be of high quality and its creator professionally trained, a less complex approach may be more suitable for environments with very large amounts of data, such as the Web. Folksonomies represent a structure to maintain metainformation that is much simpler than ontologies. The term “folksonomy”, originally coined by information architect Thomas Vander Wal, is a combination of “folk” and “taxonomy”. It has been argued, however, that the term is not accurate, as a taxonomy implies some form of hierarchy, which doesn't reflect the flat namespace used in social classification. Another term for folksonomy that will be used interchangeably is “collaborative tagging”.

3.4.1 The Purpose of Folksonomies

A folksonomy emerges when metadata creation is done by various users that may or may not be responsible for the content they annotate. All a user has to do is describe the content of a document with an arbitrary number of keywords, so-called “tags”. Usually, there is no restrictions as to what specific tags can be defined and no relations have to be established between terms. This simplicity sets a very low entry barrier, as no specific knowledge other than about the content of the tagged document is needed to participate. The motivation of a user to classify a document in such a way may be selfish

or altruistic; tags can either be used to organize some content for oneself, or to index it for other users. Nonetheless, the benefit of collaborative tagging is always general. Folksonomies encountered a great success, especially as the Web 2.0's center of attention lies on social networking, content sharing and generally higher user implication. Its simplicity allows to classify a great number of information with little overhead.

As already mentioned, folksonomies are, strictly speaking, not taxonomies, a tool to categorize documents in a hierarchical structure, but rather a classification system with a flat hierarchy of tags. A folksonomy develops organically when the framework for restriction-free tagging is provided in a social environment. This very lack of constraints is the reason folksonomies gained so much popularity, but also their main weakness, as will be seen in the next section.

3.4.2 Strengths and Weaknesses of Folksonomies

Evidently, folksonomies' simplicity is their main advantage. Whereas in the past, creating metadata was restricted to content authors or other professionals, social tagging allows everyone to participate in classifying documents. Such a decentralized classification is precisely what makes it possible to share content on Web sites like YouTube or Flickr. Without it, the costly process of assigning the resources keywords would need to be done centrally, a task that would require an understanding of the documents' content and could not be done automatically. This reveals another major strength of folksonomies. Namely, the fact that the tagging is done by specifically the people who are the most engaged with the corresponding content and who have the greatest interest in documents being labeled correctly. This allows for metadata to be potentially more precise and comprehensive. On the other hand, this also yields the risk that the metadata is biased, since every user will have a different view of what a document is about. Furthermore, the lack of a controlled vocabulary as well as any kind of structure within the folksonomy results in many possible ambiguities. Homonyms, which are same words with different meanings (for example “set”), and synonyms, different words describing the same thing (like “car” and “automobile”), are such sources of confusion. There are also the questions of pluralization (“book” or “books”?) and the use of inflected terms (“programmed” instead of “program”) and acronyms (“owl” might be a bird or refer to the Web Ontology Language, since letter case is usually disregarded). In addition, the problem of how to deal with multiple words (such as “New York”) and whether to allow spaces in tag names or not is often not consistently defined. There is

also no way to control wrong or deliberately misleading tags. All these factors can make it more difficult to retrieve relevant information and are the price to pay for the simplicity of folksonomies, as a predefined vocabulary would be harmful to their expressiveness and flexibility. A possible solution in order to introduce at least some consistency is to propose to a user who is about to tag a document a set of similar tags that are already in use.

A further point in favor of folksonomies is that even though individual tags might be flawed, the network effect ensures a good overall classification quality. More appropriate and fitting tags will convey a resource more traffic and will thus be sought after by taggers. In relation with this is the direct reflection of user vocabulary in a highly dynamic fashion, as for instance the use of recent terms or newborn words that have been coined by the tagging community. When searching for a document, a user will find metadata to be defined with the same set of words he employs. An interesting analogy has been made by [Merholz 2004], who argues that folksonomies are similar to foot-worn paths in a landscape. Over time, trails appear along the most frequently used routes, creating a user-defined network of paths guaranteeing optimality and adjustment to user preferences. The same holds for folksonomies, which grow organically according to user needs. Once distinct paths are established, it is conceivable to define a controlled vocabulary based on the one formed by the community.

3.4.3 Folksonomies in Usage

Folksonomies can be found mostly on Web sites that center around having its users share some content. These are usually open social environments, where all a user has to do to participate is to register with an e-mail address and a user name. A lot of the services that feature Web 2.0 attributes actively encourage collaborative tagging. There are two main types of applications of folksonomies. One of them are social sharing platforms for resources such as still images, text or video documents. The most popular of this kind of Web sites that feature social tagging include Flickr (photograph sharing), YouTube (video sharing) and generally blogs for textual content but also other media formats, although blogs are different, since it is usually only possible to query a single blog's folksonomy at a time (except with a blog search engine). Once a registered user has uploaded a document to such a platform, he has the possibility to add arbitrary tags to it. Flickr also allows to specify tags later on and by users other than the “uploader”. Furthermore, a user is free to decide if he wants to share his photographs with other users. Video sharing platforms, however, typically serve the purpose of having a video seen

by a large public, while saving on costs like hosting the video on a Web site or the programming knowledge required to do so. Strictly speaking, despite its use of user-created tags to describe the documents, according to [Vander Wal 2005] classification systems such as YouTube's aren't exactly folksonomies, as the tagging is open only to the publisher and merely reflects his vocabulary and not the community's. Also, the YouTube search engine relies more on clues like video title, number of views or user ratings than on tags.

An application that really harnesses the power of folksonomy tagging is social bookmarking. Here, the main goal pursued is a totally different one: the organization of documents for the user's personal benefit. Services such as Delicious, BibSonomy and the now defunct Furl and My Web provide a framework for users to submit a Web link and to tag it with any vocabulary. The tags, however, are proper to the user, not the document. This means that, after a user has bookmarked a Web page, his bookmarks can be seen by other users, who in turn can bookmark the page for themselves with their own tags. Hence, a same tag can be applied to a document multiple times, but is never a permanent part of the associated resource. In such a system, the strengths of folksonomy tagging fully come into effect, as it becomes possible to rank search query results by the number of times a tag has been used for a document or to rapidly get an overview of a resource's content. Collectively, the taggers create invaluable precise meta-information, while still potentially addressing different user vocabularies. While found mostly on social bookmarking platforms, this type of folksonomy could probably be applied on any kind of resource sharing site, with registered users storing and tagging resources on their account

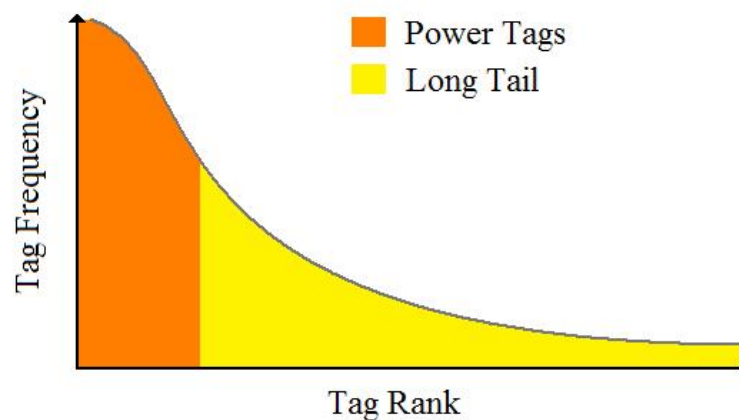


Fig.3.3: Distribution curve of tag frequency

page. In the folksonomy classification scheme explained in [Vander Wal 2005], the author refers to this as a “broad folksonomy”, because there can be several sets of tags for a document, allowing a much broader vocabulary. This results in a power-law distribution of the tags, as shown in Fig. 3.3, with tags ordered according to their frequency. The majority of users will define a document with a small set of tags, the orange part of the graph. These are called “power tags” and are the most popular and likely the most precise terms to describe the content of a resource. The tags at the yellow colored right end of the curve, the “long tail”, reflect the vocabulary of a minority of taggers. This illustrates the benefit of a broad folksonomies: for one, the ability to democratically define a set of the most fitting tags and to use it for relevance ranking. Another way to make use of power tags is to visualize them with a tag cloud, a list of terms with their importance highlighted typically through font size. On the other end of the distribution is the possibility to address many different user groups with distinct vocabularies or even languages.

This possibility does not exist in narrow folksonomies, like the one used by Flickr is. While there is only one, often limited set of tags per resource, such a classification can still add value for otherwise non-textual documents, even though no weighting or other distinction of the tags is done. However, knowing who and how many users defined a specific tag is an important requirement for the advantages of folksonomies to take effect.

Part II:
Harnessing Folksonomies in Theory
and Practice

4 Harnessing Folksonomies

4.1 Introduction

After introducing all the relevant theoretical elements, it is now time to shift the attention to the central point of interest of this paper, the practical part. At its core is the goal of exploring the possibilities to harness folksonomies and to exploit their advantages over other types of metainformation. This will be done by picking the categorization of collaborative tagging back up from Chap. 3 and looking at a narrow and a broad folksonomy in detail. With this starting point, the discussion will center around the question of how the defined types of folksonomies can be used by a Web crawler, which in turn will lay the foundation for the next chapter, the implementation of such a crawler. Exactly why the question of harnessing the power of folksonomies is relevant and what the opportunities that reside in it are, is the subject of the next section.

4.2 Folksonomies and Web Crawlers

Before actually tackling the central issue of this chapter, it would seem appropriate to make clear what in fact the relevance of making use of collaboratively created tags with a Web crawler is. The question of why this actually is a problem will also be addressed, as there don't seem to be any Web search engines that acknowledge folksonomies.

4.2.1 Identification of Tags

As shown thoroughly in Chap. 2, general-purpose Web search engines record all or most of the words in a Web page in order to index it. To determine which extracted words are the most important, that is, which ones are likely to be the best suited to describe the page at hand and thus need to be assigned a higher weight by the indexer, usually markup information is taken into account. For example, in HTML, headings are defined with specific tags for different importance levels ranging from `<h1>` to `<h6>`. Furthermore, the `<title>` tag can be used to name a Web page. This kind of metainformation can easily be harnessed by an search engine, since the syntactic representation is uniform and the instructions of how to handle different tags can simply be hardwired into the indexer's code. Thanks to the existing standards, processing regular Web pages is not difficult.

In the case of folksonomies however, such standards are missing. Despite the large number of Web sites relying on collaborative and other forms of tagging, there is a lack of a uniform method for its implementation. Although often sharing a mostly similar visual representation, the underlying markup varies greatly from site to site. This has for instance been pointed out by [Biglione 2005] for folksonomy menus, which are typically represented as tag clouds, but in very different ways. Apart from usually being the anchor text of a link to a page listing all the documents annotated with the same term, a tag is not different from other non-heading text, or at least not different in a consistent way. Being more of a concept stemming from practical use rather than a formalized method, folksonomy tagging is implemented in an ad-hoc fashion according to whatever goal is to be achieved. Folksonomies are being adapted to existing technologies, and not the other way around.

As a result, there is no way for a machine to tell whether it is dealing with a tag or regular text. Hence, a search engine indexer will not know better and assign to what is in fact metainformation the same importance value as to normal words in the document. Strictly speaking, this isn't a problem, since the tags may still be used for indexing without the search engine being able to identify them as such. Rather, it is a missed opportunity to enhance the precision of Web search engines by including additional metadata and harnessing all the advantages of folksonomies, as will be discussed in the next section.

4.2.2 The Use of Folksonomy Harnessing

The dominance of folksonomy tags over other forms of metainformation for Web pages should be evident at this stage. To recapitulate Sect. 3.4.2, not only does a folksonomy represent a simple, easily accessible classification scheme, but also a highly precise and dynamic set of metadata, provided enough participants exist to set a network effect into motion and to make up for the lack of structure and controlled vocabulary. From the viewpoint of Web search engines, further benefits emerge. The huge number of Web documents published by users, for instance, represent a big challenge for search engine indexers, particularly non-textual resources such as pictures and videos, due to the fact their content cannot be interpreted by machines, a challenge that can be tackled by the use of tags. Even in the case of a written document, collaboratively created user tags might capture its essence more exactly than a keyword-based index with the added advantage of including terms that don't appear in the text but are equally well suited to describe the content, like synonyms and other contextual information.

Tags are not only used on platforms for user-created resources, but also to annotate Web pages in

general. Despite not actually representing a folksonomy, the tags created by the author of a blog to classify his entries can also be exploited for indexing by an external entity, such as the blog search engine Technorati (www.technorati.com). More interesting, however, is the collaborative tagging promoted by social bookmarking sites. Additionally to the previously mentioned benefits of having multiple users create metadata for a document, social bookmarking could be potentially useful to a Web search engine for the ranking of Web pages. Just like a hypertextual reference to a URL is an assertion of authority for link analysis algorithms, the creation and the tagging of a bookmark is an equivalent statement about the quality of a page. A Web search engine based solely on tag and bookmarking frequencies across different platforms is then conceivable. Of course, the overwhelming majority of Web pages is not referenced on social bookmarking sites, but a similar problems occurs for regular search engines, as certain portions of the Web graph are less likely to be indexed. Some more ways to harness folksonomies will be explored in Sect. 4.4 after a brief discussion of the types of collaborative tagging.

4.3 Narrow and Broad Folksonomies in Practice

Using the classification of folksonomies introduced by [Vander Wal 2005] and explained in Sect. 3.4.3, which separates them into broad and narrow ones, this section is meant to review practical examples for both types of collaborative tagging. The purpose of this is to acquire an understanding of the way folksonomies work in reality and to gain some insights on potential methods to harness them.

4.3.1 Definition of Folksonomy Revisited

There generally appears to exist some confusion over the exact definition of a folksonomy, or more precisely over the boundaries of narrow folksonomies, so it would seem to be appropriate to clarify the situation beforehand. Collaborative tagging, per definition, requires the possibility for multiple users to tag the same item. This condition holds for both types of folksonomies described in Sect. 3.4.3. The mistake has been made to center the classification of folksonomies around this very criteria of single vs. multiple users tagging a document, when in fact tagging system that only allow for one user, usually the publisher, to annotate a resource are neither narrow nor broad folksonomies. The actual criteria determining the membership to a narrow or a broad folksonomies is whether the set of tags for a corresponding document is singular or plural, i.e., whether there is one for all the users or a separate one for each tagger.

4.3.2 Practical Example of a Narrow Folksonomy

By far the best-known and most frequently cited instance of a narrow folksonomy is the photo- and video-sharing platform Flickr. As a matter of fact, Flickr is actually only a restricted folksonomy, since tagging of other users' photographs and videos is only allowed with the publisher's permission. Still, it is a sufficiently well-suited example of a narrow folksonomy implementation. Tagging is mostly done by users for their own documents with the purpose of either managing collections of them, annotating them for other individuals to find or a combination of both. Defining tags is optional and uncontrolled. However, a dominant folksonomy vocabulary is likely to emerge, since if photographs are tagged with the intention of being shared with a large public, users will adapt their tags to already established ones. As on any on-line community platform, the creation of an account is the basic requirement to participate. Tagging on Flickr is quite simple: after uploading a file in one of the many accepted formats, and which may for paying subscribers also be a video of up to 90 seconds, the user is presented with the options of adding a title, a description and up to 75 tags. Individual as well as multiple words (specified within quotation marks) are accepted tags. On the profile page, the user then has various options to define specific permissions related to his profile and photos. The most interesting feature in this paper's context is the possibility to let other users tag one's documents. This permission can be extended to all the users on Flickr or just a restricted set of them, such as contacts. The defined individuals can then simply add any tag they wish in the same manner as for their own photographs. Tags can also be removed, either by the person who added them or by the corresponding image's publisher.

Evidently, the strengths of folksonomy tagging only come to play when adding tags for a document that is permitted to be viewed by everyone is open to at least a few other users. Only then can these keywords be trusted to render an at least appropriate content description. A small number of taggers are enough to do so sufficiently well. But in order to enhance the vocabulary, a larger amount of taggers are necessary, which is barely ever the case in narrow folksonomies. Another problem emerging with narrow folksonomies is its inability to prioritize tags. With broad folksonomies, tag frequency is a clear indicator to identify the most relevant ones, a possibility Flickr doesn't possess. This raises the question how the results of the in-site search engine are ranked. Flickr's search function in fact even features three sorting criteria: relevance, recency and interestingness. While the relevance ranking seems to rely mainly on meta-information supplied by the publisher, such as title, description and obviously tags, and the recency ranking of course on the date of publication, the interestingness ranking for some part takes

into account how much attention a document is given to. This includes the number of tags, annotations, additions to lists of favorites and comments written to a specific resource by other users, and can be regarded as the approximative equivalent to the number of users that create tags in a broad folksonomy. It might also be of interest to have a look at what types of keywords are being used. On Flickr, as it is typical for sites that feature collaborative tagging, the most popular tags are visualized as a tag cloud. A screenshot of it as it looked like in October 2009 is shown in Fig.4.1.

All time most popular tags

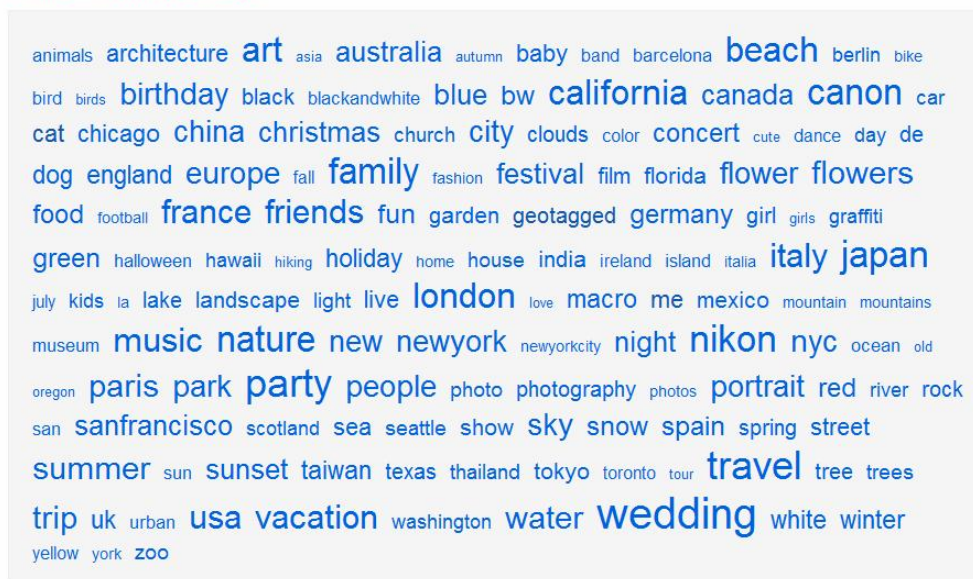


Fig.4.1: Screenshot of Flickr's most popular tags represented in a tag cloud [Flickr 2009]

The most common tags are by far the ones determining the location where the photograph was taken, be it a country, a particular city or a general landscape (“beach”, “mountain” or even simply “nature”). This category partly overlaps with the tags that describe the subject of the picture (“park”) or the setting (“sunset” or “winter”). Another common subject seem to be pets and plants. This list also reflects the most likely events to generate photo opportunities, namely social gatherings (“party”, “wedding”, “family”) and holidays (“vacation”). From the folksonomy point of view, two particularly interesting phenomena can be observe in Flickr's tag could. One thing that sticks out is the evident lack of controlled vocabulary. There are several different tags for photographs of New York City, as no consensus has been found by the community whether to put it into a single tag (white spaces are ignored in the tag could), split it up into separate tags (possibly due to users ignoring how to define

multiple word tags) or to use the acronym “NYC”. A similar disagreement over the singular/plural form of tags (“tree”/”trees” and “mountain”/”mountains”) appears. A second noteworthy phenomenon is the emergence of keywords specific to a certain group of users. Many terms related to photography can be spotted, such as tags for genres and techniques (“portrait”, “macro”, “blackandwhite”) or for camera brands (“nikon”, “canon”). Another interesting entry is “geotagged”. It refers to the presence of a special kind of tags that indicate the coordinates of the photo location in the form of so-called “machine tags”. Apart from locations, machine tags allow to associate a document with a specific event, user or basically any thinkable object. The syntax for a machine tag is *namespace:predicate=value*, which is why it is sometimes also referred to as “triple tag”. For example, a picture could be annotated with the tag *bird:species=sparrow* to define its subject or *upcoming:event=1234* to associate a photograph with an event as defined by its identification number on the event calendar site Upcoming (upcoming.yahoo.com). As their name indicates, machine tags can be processed automatically, which allows for general searches, such as for all the photos in a given namespace, or also more refined ones, like images of a specific event. Machine tags are defined exactly like regular tags and their namespace is open, making them a sort of hybrid between folksonomy tagging and semantic annotation.

In a comparison with other Web pages allowing user tagging, it becomes obvious that the tags are implemented and listed differently on each site. One common element for most such sites is that the tags are represented as anchor text for links to a page listing search results for the corresponding keyword. But aside from that, the lack of standardization, as well as site-specific needs contribute to the wide array of different means of using tags.

Narrow folksonomies provide a good way to create metadata for objects such as photographs and videos. However, not only the often scarce number of tags for a particular object, but also the lack of information about the use of those tags make it very difficult for an outside entity without contextual knowledge to estimate the relevance and authority value for this document.

4.3.3 Practical Example of a Broad Folksonomy

Just as Flickr is for narrow folksonomies, the social bookmarking site Delicious is the example of a broad folksonomies most often referred to. The main motivation for the user is personal: to bookmark relevant Web pages in order to retrieve them later, through a personalized classification and independently from a browser. As on Flickr, a registration is the only mandatory requirement to

participate, and the tagging, also devoid of any restrictions on the vocabulary, is optional. Contrary to Flickr, however, a resource isn't shared by uploading it to the site's server, but simply by specifying a link to it. Hence, a single link can be tagged several times by different users, each one adding his own set of tags. This is a defining characteristic of a broad folksonomy, as it has been discussed in Sect. 3.4.3, with the tags covering different vocabularies and tending towards a power law distribution. Delicious allows to make the most out of its folksonomy by providing the option to view the most popular tags for a resource, as well as the set of keywords defined by particular users or groups of users. This device makes it possible to group tags by overall frequency or by popularity within a specific community with a possibly specific vocabulary. The same options exist the other way around, that is, finding resources for a given tag from the user's personal bookmarks or from those of a set of users, for example with the site's search function.

The number of tags that can be added to a single resource is unlimited, but defining multiple word tags isn't supported, which is a potential source of dissent within the folksonomy vocabulary. To enforce a standard tag set, Delicious suggests a few already used tags to the user about to add a bookmark. These suggestions are grouped into the most popular keywords defined by other users for the same document and “recommended tags”, the intersection of the user's personal tags with the set of tags used by the community for that resource. This helps a user to at least be consistent with his own vocabulary and, if he wishes so, also with the majority of other users. For a better management of a very large tag list, it is furthermore possible to create “tag bundles”, which are user-defined collections of tags. These can either be used to simply group similar tags together for quicker access or to filter bookmarks out that have tags belonging to a specific bundle. For example, a user may want to group bookmarks tagged with “folksonomy”, “bookmarking” and “tagging” in a tag bundle called “web2.0” and thus define a two-level hierarchical structure. While this may seem to contradict the central folksonomy concept of a flat vocabulary, such a classification only applies to the user's own tag set.

As with the Flickr example in the previous section, having a look at the most popular tags can reveal quite a lot about how Delicious is being used. Fig. 4.2 shows the bookmarking site's tag cloud, and here again, the most common tags evidently reflect an average user behavior. Many keywords, such as “software”, “tools”, “programming” or more specific ones like “javascript” and “linux” indicate an interest for technical contents. Then, a certain number of tags are genre descriptors, like for example “blog”, “video”, “tutorial” or “photography”, or define a particular subject (“music”, “art”, “food”, “technology”, etc.). The emergence of a specific vocabulary is also conspicuous on Delicious' tag

cloud. For one, the community appears to have agreed on the convention of merging multiple words into a single tag without a special sign in between, as in “webdesign” or “opensource”. With the help of the in-site search function, this phenomenon can easily be proved.

Tag Cloud: Popular

KEY: [blue tags](#) are tags you have in common with everyone else.

Sort: [Alphabetically](#) | [By size](#)

design blog video software tools music programming webdesign reference
 tutorial art web howto javascript free linux web2.0 development google inspiration
 photography news food flash css blogs education business technology travel shopping
 books mac tips politics science opensource games culture research java windows security
 internet movies online search humor funny social community fun mobile recipes cool
 marketing health php tutorials cooking resources history portfolio audio download graphics
 media library toread python photo article ruby ajax learning film maps photoshop youtube
 architecture rails computer wordpress freeware plugin home hardware firefox apple mp3
 illustration photos email twitter socialnetworking api ubuntu language database fashion osx tv
 blogging network html book typography interesting work money finance japan advertising
 productivity list recipe magazine environment webdev writing jobs 3d 2008 code guide icons
 imported images game networking diy cms videos lists wiki seo green gallery usability jquery
 microsoft tool collaboration .net privacy visualization entertainment psychology tech movie
 statistics iphone articles management phone desktop podcast math shop economics geek
 radio ebooks drupal comics people rubyonrails forum flex reviews information animation
 government browser data wikipedia hosting vim religion school wishlist realestate todo house
 literature rss fic converter streaming downloads electronics teaching interactive kids
 documentation car flickr and artist

Fig.4.2: Screenshot of the Delicious tag cloud [Delicious 2009]

By entering “tag:web” for instance, only bookmarks with this tag are returned. A quick investigation into the three most frequent multiple word tags “Web design”, “open source” and “social networking” clearly shows that a large majority put the words together, with respectively ca. 634'000, 313'000 and 156'000 bookmarks tagged that way. In comparison, there are only 241'000, 39'000 and 58'000 resources annotated with both words in separate tags¹. Interestingly, out of these users, about the half chose to also supplement the merged tag, and thus to make a conscious effort to extend their bookmarks' retrievability to different search habits. A third tagging method for multiple words is the use of an underscore character (“web_design”), but is only employed by a minority, with 35'000, 22'000 and 27'000 bookmarks for the respective keywords. What this shows is a power law

¹ Bookmarks featuring both tags were counted here. However, there is no way to prove that they are meant to be a single term, i.e., that “web” and “design” are directly related.

distributions, as there is a large number of other characters in use to separate words in a tag, such as the plus sign (“+”), the minus sign (“-”) or the period (“.”), each with significantly smaller numbers of bookmarks. Amongst the top tags also appear tags obviously used as personal reminders (“toread” and “todo”), indicating that not all tags are generally valid metadata.

Delicious doesn't support machine tags with a search functionality the way Flickr does. Tag bundles are the closest thing provided to the user to help creating a simple taxonomy, with the bundle name being the namespace, the tags as value fields and the predicate left undefined. However, Delicious automatically adds machine tags to certain kinds of resources. More specifically, bookmarks of audio, video, image and document files are annotated with a tag describing the type of media as well as the file type. A bookmark of a PDF, for example, will be added the tags “*system:media:document*” and “*system:filetype:pdf*”.

Delicious is a perfectly suited object of study for folksonomies, as it features numerous search options. Aside from the logical operators (*and*, *or*, *not*) used to combine or exclude certain keywords, searches for specific periods of time and groups of users are supported. As words are typed into the search bar, keyword suggestions are automatically provided, a functionality that further promotes the emergence of tagging conventions and power tags.

The example of Delicious shows that with a broad folksonomy, the benefits of harnessing collective knowledge are really set into motion, as multiple vocabulary sets make it possible to take tag frequency into consideration for the ranking of resources, while also allowing for more specific searches and encompassing folksonomies' lack of structure.

4.4 Methods for Harnessing Folksonomy Tags

It is now time to explore a few possible options of using the metainformation contained in folksonomy tags to index Web resources. The following section is in fact pivotal to this paper, as it builds on the topics discussed up to this point and in turn serves as the basis for the implementation of a prototype of a Web crawler. Each subsection contains the description of a method to do so, as well as some advantages and disadvantages. Again, the point is not to find ways to index specific types of resources, but to make use of the metainformation provided by user-created tags, whether it be a video, photo, text or other type of document.

4.4.1 Extraction of Formally Defined Tags

While there are no generally valid standards neither for the markup of folksonomy tags, nor concerning their namespace, some local formalizations of tag syntax do exist. For completeness' sake and to show the entire range of the potential of user-created tags, markup as well as namespace conventions will each subsequently be illustrate with an example and discussed.

A search engine that already harnesses user-created tags is Technorati, albeit it only does so for blogs, and moreover only if the blog creator agreed to some conventions. More specifically, these conventions refer to the way tags are represented in the HTML code, namely as a hyperlink with, in addition, a “tag” value for the *rel* attribute. This attribute is normally used to specify the relationship from the current page to the one referred to and has an open namespace. Although not being supported by any current browser, its value can still be picked up by a Web crawler to gather further metainformation. Another important element of Technorati's tagging convention is the URL specified with the link, which is required to end in a slash (“/”) followed by the name of the tag, but otherwise arbitrary. Usually, the blog's page of documents with this tag is used, like for example “<http://www.someblog.com/tag/cat>”, or Technorati's page for that tag. The definition of a tag might then look like this:

```
<a href="http://technorati.com/tag/cat" rel="tag">Cat</a>
```

To make the tag invisible to the blog's readers, the anchor text can simply be left blank. Exactly how Technorati notices changes on blogs or does the ranking is not relevant here. The point is to show that there are simple ways to represent tags in a uniform way and to retrieve them with a consistent mechanism, like in this instance just scanning a page for *rel="tag"* expressions, extracting the keywords and using them for indexing.

Even though the present example does in fact not count as a folksonomy, as only content creators specify tags, its design could very well also be applied to collaborative tags. The mechanism to automatically convert user-defined keywords into the proper syntax would have to be provided by the Web site, which, from a technical viewpoint, wouldn't be a big deal, since tags usually already are links to a page listing other documents for the same tag. There would, however, need to be a motivation for resource sharing platforms to implement this change.

A formal definition of tags may also apply to their namespace. The idea is not to restrict the folksonomy's vocabulary, but to specify a syntax to enhance the meaning of an entry. Machine tags, as seen in Sect. 4.3, are a way to create a simple taxonomy of tags and have the benefit of both enabling

automatic procession and still being user-defined from an open namespace. Thus, they represent a middle ground between unstructured folksonomy tags and machine-centered metadata. Machine tags can provide very precise results for a Web search engine. Considering the example of a user looking for photos of a specific event, machine tags would be a great solution to find relevant documents without having to rely on metainformation defined by other users and search engines. Since they are defined exactly like regular tags, they could be extracted the same way. But like for normal tags, there is no standard form to display machine tags on a Web page, using the above would however be conceivable. Due to the uncontrolled vocabulary, machine tags also deal with the problems of unclear or overlapping definitions. The network effect could here as well help to shape community conventions, as it did on Flickr. Nonetheless, machine tags are not widely used, probably because of not being as accessible to a large audience as regular tags and the difficulty to describe subjective attributes with them.

4.4.2 Harnessing the Folksonomies of Social Bookmarking Platforms

A highly interesting source of metainformation is provided by social bookmarking platforms that implement collaborative tagging. First, because they represent the only thinkable way to have a large number of people create metadata for Web resources “from the outside”, i.e., not doing so while on the corresponding site, and thus in a consistent fashion. Second, all kinds of document formats are being tagged with social bookmarking, not just a single one, as it is usual when user tagging is supported. Third and finally, because implementing broad folksonomies with multiple tag sets per resource allows them to reflect the authority of Web pages through tag frequency. This authority, however, is a different one than for example the one asserted with Google's PageRank, as it is more prone to reflect a measure of subjective interestingness than of a Web graph situation. Hence, it might be of interest for a Web search engine specialized on informational queries, and less on navigational or transactional ones (cf. Sect. 2.3.4 about query categories), to have its crawler gather tags as well as their respective frequency for specific documents. The question is then what path to use to visit bookmarks and how to extract tags and information on them.

Ideally, the data set to be crawled should be taken from multiple different social bookmarking sites in order to have enough relevant documents and user “votes” to determine this relevance. Given however the scarce number of such platforms that actually implement broad folksonomies, it would also be conceivable to include social bookmarking sites that rely on voting mechanisms to determine the popularity of bookmarks rather than on tags, sites such as Digg (digg.com) or Reddit

(www.reddit.com). This way, even with only one or two platforms with folksonomies, indexing could be done reasonably well, while additional information for ranking could be gathered from those other sites. A problem with this approach is the fact that the different communities need to have common fields of interest, or there will be too large discrepancies in popularity for a reliable ranking.

Regardless of such considerations, the question of how to design the crawling path remains. A first method would be to crawl the sites at query time, that is, index and rank URLs as a query comes in. This could be done by heading straight to the corresponding tag pages and index one URL after another as they appear there. Some sort of algorithm would have to be defined to determine the weighing of results from different platforms and with varying numbers of tags. Bookmarks could alternatively also be crawled at random and off-line, by, for instance, starting from the social bookmarking site's home page, which usually lists some of the most recently added links. Another potentially interesting starting point could be the listing of the most popular tags. Working its way down this record, a Web crawler would make sure to index URLs that best capture the communities' interests. A more general-purpose search engine could also just use popular bookmarks as seed pages for its crawl and start traversing the Web for other high authority documents from there. A further approach could be to assert authority to certain users, or some more specific combination, like defining a set of users particularly knowledgeable on a subject that might be relevant for a specialized search engine.

Either way, an implementation of any of the above methods would be rather closely coupled with the target Web sites' code and display a very restricted flexibility, for example when a Web site's representation of tags is changed, or a new source is to be included. Also, as mentioned earlier, social bookmarking sites with broad folksonomies are quite limited in number, so that a search engine for collaboratively labeled resources only adds little benefit over a direct on-site search functionality. Still, the above discussion is not only aimed at finding practical solutions to harness broad folksonomies, but also to show the possibilities connected with user tagging. Once more, the conclusion is that facing different implementations, it is difficult to extract tags in a consistent and easily extensible way. With that in mind, the following section examines a few implementation-specific methods.

4.4.3 Ad-hoc Tag Extraction

Since there is obviously no consistency in the representation of tags across different sites, the designer of a Web search engine intended to include collaboratively labeled resources might just decide to roll with it and adapt the crawling for documents and their tags to the relevant sites' specific

implementations. Upon visiting a page, a Web crawler could compare its URL to a list of sites from which it has already been “taught” to extract tags, or rather to identify which parts of the parsed HTML code to have indexed as metainformation and thus weighted more heavily. If a match is found, the tags can be extracted accordingly, otherwise, the crawler can go about processing the Web page normally. This way, any folksonomy deemed relevant by the search engine designer can be harnessed and further ones continually added. Obviously, the “teaching” part would be the most cumbersome in the implementation of such a search engine. To see how this could work, let's consider for example how tags could be extracted from a Flickr page. Either accidentally or by design, a URL with the domain name *flickr.com* comes out at the end of the Web crawler's frontier. As the page is fetched, it is quickly looked up for a match in the list of known folksonomies. Assuming a method has been defined how to extract Flickr tags, it is executed. On Flickr, tags are represented within a HTML *div* element and within this are also described in a *script* tag for some JavaScript code, as shown in Tab. 4.1. The tags

```
<script type="text/javascript">
global_photos['4033717923'].tagsA.push('powerlaw');
global_photos['4033717923'].tags_rawA.push('powerlaw');
global_photos['4033717923'].tagsA.push('curve');
global_photos['4033717923'].tags_rawA.push('curve');
                                [...]
</script>
```

Tab.4.1: Sample code for a Flickr resource [Flickr 2009 (2)]

used to label this image are “powerlaw” and “curve”. As it can be seen, both tags are used twice as variables for some functions within parentheses and single quotation marks. Disregarding their purpose, either of these functions can be used as a sort of orientation point for the indexer to find tag terms. The program simply has to be told to store the words in the brackets and pass them along to be used for indexing as folksonomies tags. Of course, for additional indexing precision other elements such as titles could also be stored. Like in the above example, an indexer could be adapted to any implementation of tags and make them accessible from the outside. Although so far only narrow folksonomies have been considered, there is no reason why such a design wouldn't be applicable to broad ones as well. Tab. 4.2 illustrates the markup for tags on Delicious. It shows how the tags (only the top two are displayed here) and their frequency for a particular bookmark are represented. Every list item (*li* element) stands for a tag, within which its name (“technology” and “news”) as well as its

frequency (2952 and 1779 respectively) are mentioned several times. After making sure a certain part of the surrounding code only appears one time in the page's entire source code, the programmer could tell the indexer to head directly to that specific piece of markup and the steps to follow from there to get to the pertinent information. In the above example, the *div* element with the value “*top-tags*” for its *id* attribute would be such an eligible code element, and the value of the anchor's *title* attribute could then be extracted and processed further.

```
<div id="top-tags" class="sidebar-list toggle on">
  <h3><span class="toggle-button yes">Top Tags</span></h3>
  <ul class="list">
    <li class="first">
      <a href="/tag/technology;_ylt=A0wNBsN_5PJkrAMBtKFvRh54;_ylv=3" title="technology (2952)" class="showTag"> <span class="m" title="technology (2952)">technology<em>2952</em></span></a>
    </li>
    <li class="">
      <a href="/tag/news;_ylt=A0wNBsN_5PJkrAMBtKFvRh54;_ylv=3" title="news (1779)" class="showTag"><span class="m" title="news (1779)">news<em>1779</em></span></a>
    </li>
    [...]
  </ul> <div class="clr"></div>
</div>
```

Tab.4.2: Sample code of Delicious tags [Delicious 2009 (2)]

It is obvious that the concept of manually adding and whenever necessary rewriting the code for each Web site to be crawled for folksonomy tags is rather tedious and doesn't scale too well. Without some folksonomy standard, extracting the tags in a uniform way doesn't seem to be a much easier alternative. However, this is precisely what the following method is attempting to do.

4.4.4 Semi-Automatic Tag Identification

There would be no need to adapt the crawler code each time new folksonomies are to be added if there were a common representation for them and a Web crawler could collect and identify tags just as it does with other HTML elements, such as page titles, headings or anchor texts. Since this is not case, another way has to be found for the crawler to still be able to automatically recognize folksonomy tags.

Facing the fact that implementing a program that can tell relevant from non-relevant meta-information autonomously and in a spontaneous fashion appears nearly impossible, at least some amount of human involvement seems inevitable. That is why the following proposed method involves the user in the process of defining what relevant information to extract, without him having to rewrite any code.

While lacking standards regarding their markup, user-created tags still mostly share some common features. For one, the keywords for a corresponding resource can typically be found on the same Web page at some peripheral position. Often, the tags also double as anchor text for the user to find documents labeled with the same word. A look at the source code reveals that on a lot of occurrences, the tags are located inside a *div* element, which in HTML is a container element that defines a specific division of the document and can be associated with CSS to create the page layout. However, this element is not necessarily used each time tags are displayed, as other ones can equally well be used. For this reason, programming the indexer to look only for *div* elements would not be sufficient, in addition to the fact that these can also serve a wide array of other purposes. In fact, relying on any specific element of markup to retrieve tags consistently is problematic, as it is bound to quickly be made useless and outdated by a differing folksonomy implementation. Evidently, the implication of the user is required in order to identify the tags. This statement might seem familiar, as it already was the motive for the method discussed in Sect. 4.4.3. However, this time the intention is, instead of having a new piece of code for every instance of a folksonomy, to be able to harness different representations of tags with the same program. For this to be possible, it needs the ability to learn in order to expand its set of harnessed folksonomies, which in turn requires human interaction. This interaction could take place through some user interface that would allow the user to specify to the program what exact part of a page is relevant, that is, either by defining the corresponding piece of code, or, with a more sophisticated application that would display the Web page like a browser, by marking the tags directly on the page. With the evident assumption that a site's tags are always represented the same way, the program could then store that information alongside with the top-level and second-level domain name of the site, or preferably even a further specified URL in order to narrow down the set of pages with the same specific implementation of tags even better and to avoid looking for tags on pages with content other than shared resources, like the FAQ or the site's home page. Also, multiple tag representations on a single site could then theoretically be addressed. The exact information that the program would store for a given set of URLs would be the specific HTML element containing the tags, that is, the type of element and some unique property of that element, such as the value of a *div* element's *id* or *title* field.

When gathering folksonomy tags from a page, all that is left for the crawler to do, is to find that specific HTML element, extract the output text within the element, which is the text located outside of any HTML tag, and pass it on to be used accordingly for indexing. Doing the same with tags that come with additional properties, such as tag author or frequency, as could be the case in a broad folksonomy, would require a more refined adaptation to the corresponding page's source code and possibly rule out automatic procession, but regular tags can just be processed the way they are, since they can still be tokenized later.

Having the tags be identified by the user guarantees that the tags can be found unmistakably, while the crawler's ability to adapt to different implementations allows its code to be totally independent, apart from the HTML specification and the assumption that tags can always be found grouped together in a common HTML element. Of course, many implementation-related issues remain for such a Web crawler, but these will be addressed in Chap. 5, since this is the chosen method that will be translated into action with a prototype.

4.5 Indexing and Ranking with Folksonomy Tags

The discussion of how folksonomy tags can be used for indexing and ranking of their corresponding documents deserves a section of its own, since both of these processes are basically independent of the way the tags have been collected. The only information these modules actually need are text segment, such as tag keywords, and the page element they were extracted from. To harness the advantages of broad folksonomies, however, further information evidently also need to be provided. This case will also briefly be addressed.

4.5.1 Parsing Folksonomy Tags

After the tags have been extracted, it may seem that there isn't much else that needs to be done before using them for indexing. In fact, forcing them into a controlled vocabulary appears to go against what folksonomies stand for, namely user-created metainformation with an open namespace, so this would be a point in favor of processing the tags the way they are. On the other hand, a minimal amount of control does seem necessary in order not to litter the index with countless long tail tags and considering the great number of potential tags that express the same keywords. While spell checking would be likely to discard terms proper to some emerging vocabulary, removing certain characters, such as ones used to separate words (cf. Sect. 4.3.3) would yield the benefit of collapsing different tags with

identical words into a single term in the index. Then again, the question is raised of how to deal with whitespaces, since those are also often allowed. Breaking terms separated by a whitespace apart would make them easier to match with corresponding query keywords, which would most likely also be mentioned separately. However, multiple word tags with special characters as separators would then also have to be split up. The problem is that such special tokens aren't necessarily always meant as word separators and thus can't just mechanically be interpreted as equivalent to a whitespace. If keywords are merged together, that is, all special characters including whitespaces are being removed, the question emerges how the program is supposed to be able to tell at query time for which compound tags out of all the possible combinations of query terms to return results.

Concerning the far-out long tail tags, which are rarely ever relevant to any group of users, the ideal solution would obviously be to not take them into consideration for indexing. But yet again, no heuristic method could conclusively determine the relevance of a tag with no contextual information. A solution to this problem can only be given in a broad folksonomy, which provides a reliable measure for a tag's suitability. Even without that, certain tags can be removed directly in good conscience, mainly, as in full-text indexing, stopwords (“a”, “the”, “and”, etc.) or other common not content-related tags. Stemming (converting words to a canonical form) might also come in handy to group certain terms, without reducing the user vocabulary in any way. Often, taggers take care of that by themselves, by adding several tags with a different form of the same term in order to boost traffic to the annotated document. Finally, the tags should also be formatted to a common case lettering. However, this is a restriction already enforced by most, if not all, tag namespaces, so it probably doesn't even need to be done by the parser.

4.5.2 Indexing with Folksonomy Tags

Contrary to full-text indexing, setting up an index with user-created tags is much less complex in each step of the way, as, once the program has learned how to do so, tags are well-defined text elements and usually small in number. Also, less parsing has to be done, since stopwords are rarely used and meaningless tags cannot be identified either way. As a consequence, when indexing with folksonomy tags, Web pages are processed much faster and the resulting index more limited. Coming out of the parsing module, each tag is equally important, that is, there is no discrimination regarding the origin of the indexed word, like there may be on a regular page with titles, headings and highlighted text. While this is case for narrow folksonomies, where the index wouldn't even require a field to store a weight

value for the tags, in broad folksonomies, as seen previously, a measure to base the weighting of tags on is provided with the indication of their frequencies. Even without this information, the weighting of extracted words could be done by including some complimentary page content. Although folksonomy tags are usually found on pages with little other descriptive text, hence their need for user tagging, extracting other elements aside from the tags, like the title of a resource or some content description, could add further information on a document to the index that may have been omitted by the tagger. In the case that no additional keywords can be gathered from these other sources, that is, some word is mentioned more than once, duplicate ones could simply be discarded and the higher weighted one kept in the index. It could actually even be argued that keywords extracted from a document title should be attributed a higher weight, since tags are more prone to be the object of false indications, also known as spamdexing, and it is difficult to assess the suitability of each tag in a narrow folksonomy, while the words appearing in the title are generally more likely to be of relevance.

Another interesting option to increase the quality of a document's index is the use of machine tags. A possible way to harness those could be to implement three hierarchy levels for the index, one for the namespace, one for the predicate and finally the one for the values, which is the list that would contain the URLs. The upper two would simply refer to an entry in the index below itself, the predicate index would for instance contain (*predicate, value*) pairs. Such a design would enable both general search queries, such as for all the documents with tags of a certain namespace, as well as more refined ones, like for a specific value.

Folksonomy tags provide many advantages, but they may often not be expressive enough for an index to be based solely on them. This means that aside from tags, a specialized crawler should ideally also be able to gather other well-defined descriptive content, especially with narrow folksonomies.

4.5.3 Ranking Documents with Folksonomy Tags

When ranking a Web page, it isn't of particular importance what kind of document it displays. In fact, any page that has been indexed with folksonomy tags could be ranked with every conventional algorithm, like for example link analysis. A Web page that is asserted a high authority by other Web pages by derivation should also have a high authority for a specific Web community and be particularly relevant for the subject it treats. However, the goal here is to discuss what the metainformation from folksonomies can bring in to determine how well a page matches the search query keywords and how high it has to be ranked.

A reason against using a link analysis algorithm to rank the results of a folksonomy-centric crawler is that the purpose is also to promote new interesting resources with the help of collaborative tags, and not ones that already have an established authority with conventional Web search engines, that is, to provide a different kind of service to the user. But the question is then how to base the ranking merely on tags and possibly on document titles, two elements that are not centrally supervised and under the sole control of their authors. Once more, this exposes an important weakness of narrow folksonomies, which is their inability to give further meaning to their metainformation, some kind of “meta-metainformation”. Due to the tags' flat hierarchy, it is difficult to reliably establish which tags most accurately describe the resource's content, so all the keywords would have to be treated equally, even though in reality it is likely that variable degrees of quality exist among the defined tags. A potential solution for this would be to take the order of the tags into consideration, with the implicit assumptions that the labeling user specified them in the same order as they are displayed and that the first ones that came to his mind are more general and suitable keywords. Both of these assumptions are likely to be wrong and even if they weren't, the question would remain where to draw the line between presumably relevant keywords and irrelevant “filler tags”, or whether to make it a gradual transition.

The solution chosen by most sites with user tagging for their own search functionality is to assign metainformation only a marginal importance and to focus the ranking on the resources' popularity, as for instance expressed by view counts, user ratings or number of comments. Such a design is very appropriate if the intention of the users making a search query is expected to be of the navigational type, that is, to find a specific document or set of documents. But a Web search engine for collaboratively tagged resources would be much more suitable for informational queries, that is, queries aimed at finding any resource with a potentially interesting content. For this reason, the question of determining the appropriateness of particular tags could simply be left unanswered and the ranking of pages done in function of keyword occurrence in tags, titles or other page elements deemed useful. The relevance of search queries' top results would thus not be guaranteed, which however is acceptable, as the user would most likely anyway be inclined to browse himself through the returned result pages.

With broad folksonomies, the situation is a completely different one. The tag frequency for a particular resource is a simple and measurable indication of how well a keyword describes its content. Ranking documents can just be done by considering the number of occurrences of tags matching the query keywords. With multiple search terms, many variations of algorithms to determine a rank value are

conceivable, like only adding up the frequencies of the matching tags, or allowing the user to have some search terms weighted more heavily and other ones excluded. Like with narrow folksonomies, further elements can also be taken into account, such as title, description or number of users that have tagged a page. Naturally, when searching through a set of documents labeled in a broad folksonomy and if tag frequency is used to rank the results, the focus shifts, intentionally or not, towards more authoritative results than when it is harder to assert a resource's quality, as with narrow folksonomies. Still, the fact that a particular Web page is very popular with a specific community, which in fact only represents a tiny part of the entirety of Web users, does not necessarily express a high authority in the Web graph, so a search engine for broad folksonomies still has a somewhat different angle than traditional ones.

Ranking Web search engine results in an optimal way is a difficult problem and it comes as no wonder that the algorithms to do so are kept as trade secrets by commercial search engines. Basing the relevance ranking on folksonomy tags alone, without any contextual information, is not possible. Two possibilities exist to remedy this problem. For one, the inclusion of other elements traditionally used by search engines, which, however, may still not be a sufficient relevance indicator. The second one is to compensate the fact that relevant search results are likely to be more scattered across the returned set by displaying them in such a fashion that the “good” ones can be retrieved more easily, for example by clustering the results, or providing a sneak preview of the page.

5 A Web Crawler Prototype

5.1 Introduction

In order to further examine how the extraction of folksonomy tags can be performed with a Web crawler, this chapter will describe the implementation of a prototype of such a crawler. A particular method out of the ones explored in the previous chapter has been translated into action with Java code. Before discussing the actual implementation itself, the next section will define what precisely the prototype is meant to do, as well as the elements that lie beyond its scope, and point out what particular designs for some of the crawler's elements' implementation, such as the frontier or the parser, have been chosen. Finally, the prototype's performance will briefly be evaluated.

5.2 Crawler Design

The crawler prototype discussed in this chapter is intended solely for demonstration purposes, that is, to display one possibility to gather user-created tags in a semi-automated fashion. Its functionalities and scope are adapted accordingly, which means a full-fledged Web search engine with corresponding features is not to be expected, but rather a small-scale, sequential program that is precisely designed to suit the present needs. The following section explains how its functionalities are implemented to do so. In addition, a user manual appended to this paper explains how to operate the program.

5.2.1 The Prototype's Scope

A great deal of what a Web search engine does and what its components are has been discussed in Chap. 1. But not nearly all of these functionalities are implemented by the prototype, neither does it address all types of folksonomies, like for example broad ones. While being referred to as a Web crawler in this paper, the prototype, however, also includes activities that would normally go beyond the set of tasks proper to a typical crawler.

The primary purpose of the prototype is to implement a method to extract folksonomy tags, to show how it can be done. The components that it includes are designed with the idea in mind, that this method can be applied with an otherwise normal Web search engine, while only focusing on tag extraction in the prototype. For instance, the frontier is designed in such a way, that URLs of

“acknowledged” folksonomy sites are pushed to the front of the queue and if other pages are crawled, they are only harvested for links and not indexed. Hence, the precision of the included search functionality is somewhat limited and dependent of the number of pages crawled and keywords extracted.

A second major goal is for the prototype to be able to include new representations of tags while not being dependent of already known ones because of a code too closely tied to some tags' markup. This is done by letting the user specify at run time what tag pages the program is to index and where to extract corresponding meta-information. However, these specifications are not stored beyond execution time and there will also be no functionality provided to load previously saved ones.

The prototype also allows the user to specify a set of seed pages, if possible close to promising resources, to start the crawl, as well as the location of the tags that are to be extracted. When crawling the page, the program can then use this information to pick up keywords for indexing. The page's title, that is, the text within the `<title>` HTML tag, is always extracted automatically.

How the particular components of the prototype are specifically designed is subsequently discussed.

5.2.2 Fetching and Crawling Web Pages

The first thing a Web crawler needs to be able to do is to fetch Web pages. Luckily, the `java.net` library provides all the required methods for that, so barely any custom code needed to be written here. The method `fetch()` downloads a page as a `String` type variable and returns an error message if the page could not be fetched, either because of a non-valid URL or because the connection couldn't be established. While of central importance to the prototype, there is not much else of relevance to be said about page fetching.

A much more interesting part of the code are the methods defined in the `Crawling` class. Also, several important variables are specified in it. One of those is the frontier, or rather the frontiers, since, in order to easier distinguish between “tag pages”, that is, pages the prototype knows how to extract tags from, and other pages, two frontiers exist. Both are implemented as `ArrayLists` of `URL` variables, but the second one, holding pages with no tags, is only used if the first one, holding the tag pages, is empty, in order for the crawl to be able to continue. The frontiers' size can be defined by the user, as well, in fact, as the crawl's length. The two major methods in this class are `crawl()` and `crawlPage()`. While the first one manages the crawling in general, `crawlPage()` handles the crawling of particular pages and does all the important method calls. After issuing a call to fetch a page's content, it instantiates an object of

the custom created *WebPage* class, whose purpose is to hold certain information on a Web page that will be relevant in a later stage, and then calls various methods, whose nature will also be discussed later. A list of URLs that have been crawled is maintained in order to keep track of the number of visited links, but also to avoid revisiting a page. *crawlPage()* itself is called in *crawl()*, the prototype's most crucial piece of code, which precisely defines which page is crawled at which point. Obviously, the crawler has to start with some seed pages, since initially the frontiers will be empty. The seed pages are left to the user to define, since these may depend on the defined tag pages. As a seed page is crawled, links are gathered in the frontiers and ideally also keywords. If the latter is not the case, the second frontier is crawled until the first one with the tag pages is no longer empty. When multiple seed pages are specified, the crawling is divided equally among them. If, for example, three seed pages are given, once a third of the crawl length is reached, a new crawl is started from the second seed page. Since the frontiers are simply implemented as FIFO queues, they are re-initialized at that point, since otherwise it would take too long for the newly gathered links to reach the front of the queue, as all the previously collected URLs in the frontier would be visited first and thus restarting the crawl from another seed page rendered useless.

The last interesting method in the *Crawling* class is the one in charge of taking the *robots.txt* indications (cf. Sect. 2.4.4) into account, although this is done only superficially. If such a file can be found, it is parsed for the string “*User-agent: **” and the pages mentioned in the following *Disallow* fields added to the list of already crawled pages in order to make sure they are not visited.

5.2.3 Web Page Parsing and Indexing

The methods described in this section are all, directly or indirectly, called by *crawlPage()*. Since all the methods are somehow linked between each other, the sequence of sections in this paper about the code's elements tries to follow its execution sequence, except for the GUI, which starts it all off, but will be explained last. So evidently, after fetching and crawling pages, the appropriate actions must be performed on them, which are the extraction of links and possibly meta-information, and their indexing. Each page that is crawled, be it a tag page or not, is scanned for links. Those can be found by looking for *href* fields within HTML anchor tags (*<a>*). What the code basically does, is chop up the page's source code until only the URL text is left. Extracting a page's title is even easier, since there is only one of it. As for extracting tags, this is only done on confirmed tag pages, so there needs to be a mechanism to identify these. A set of tag pages is always defined by the user, who specifies a URL and

the exact location in the source code where the tags can be found (more on that in Sect. 5.2.4). If a Web page's URL matches a tag page entry, tags and title will be attempted to be extracted from it. If it nonetheless is a page that is not meant to be indexed, this mishap will be noticed in time. However, in order to improve the prototype's performance, the specified URL for a set of tag pages should be as precise as possible and only include pages with index-worthy content.

Tag extraction, on the other hand, is always precisely defined, since the HTML element containing the keywords to be indexed is unique. To find this element, a helper method, *getElement()* is called, mainly to avoid making the extraction method too complex and loaded. When the user defines a tag page, an object of the class *ExtractionTemplate* is instantiated to store the URL and the location of the tags. This object, along with the crawled page's content, is the parameter that is passed on to *getElement()*. With this information, it is then easy to create a substring of the page containing only the specified HTML element. Assuming that the only text displayed within this HTML element are the tags, all that the extraction method has left to do, is get the text between a closing (“>”) and the next opening (“<”) bracket, since that is how text to be displayed by the browser is represented in HTML, and save each one in a list of *String* elements. Control is then returned to *crawlPage()*, which subsequently calls the *index()* method.

Indexing is simply done by converting the *String* objects to instances of the custom *Tag* class, which, aside from the tag name, also holds a weight value defined by the user for later ranking purposes. Before being definitely tied to their *WebPage* object, the tags are checked whether they either represent stopwords (such as “the”, “a”, “of”, etc.) or unwanted words specified by the user. This latter functionality allows to sort out terms that always appear on a set of tag pages and have also been stored as keywords because of their presence in the title or the page's list of tags. The general list of tags isn't exactly maintained as a classical inverted index, since the Web pages they belong to are only referred to within the *Tag* objects, which are also kept in an array proper to the corresponding *WebPage* object.

5.2.4 Graphical User Interface

The graphical user interface (GUI) is of no particular importance to any crawling-related task, it's just the code that runs on top of the prototype's more crucial methods and interacts with the user. It is also where the lists and objects are initialized, mostly in accordance with user-specified parameters. Upon starting the application, the initial frame lets the user choose between adding a new tag page and launching a crawl. Obviously, doing the latter without the first defies the purpose of the program, an

error that is reported to the user. In the frame to add a tag page, which can be seen in Fig. 5.2 a), the user is required to specify a URL and the source code location of the tags, that is, the HTML tag, for example “<div>”, and a field to identify it amongst all the other HTML tags of the same type, like for instance “id=’tags’”. It could also be of use to indicate what type of resources the mentioned set of tag pages hosts and the keywords to discard from indexing. The *Stopwords* field is however not mandatory. In order to make the program simpler and quicker to use, a few tag pages have been predefined, so the user only has to select, for example, Flickr from the spinner element, and the required specifications are added automatically. Any number of different tag pages can be defined, although after each one is added, the user returns to the application's start frame. Unless some problem occurs, an *ExtractionTemplate* object is created upon clicking the *Add Tag Page* button. An input/output exception might be caught should the Internet connection be down. The reason why this is already of importance at this point is because a new tag page's site is immediately checked for prohibited pages by fetching the *robots.txt* file.

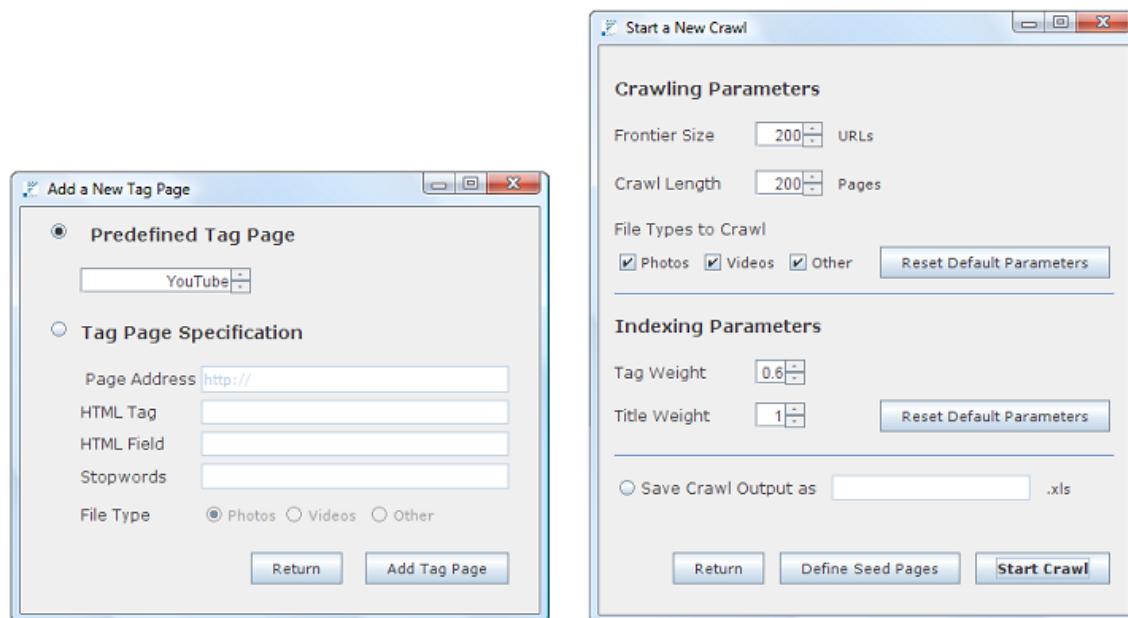


Fig.5.1: The prototype's frame to a) add a tag page and b) start a crawl

The other button on the start frame, *Start New Crawl*, leads to the one shown in Fig. 5.2 b). The crawl path and length is entirely up to the user to define. The frontier size can be chosen to be rather large, in order to have a breadth-first crawl, or can be made smaller if a more focused crawl path is preferred. The trade-off when setting the crawl length is obviously execution time versus scope of indexed

documents, even though the latter is never going to be very large, as the maximum crawl length is set at 200 pages, because the program is otherwise likely to cause the Java heap to overflow. Another important factor for the quality of a crawl are the seed pages. The closer they are located to the actually relevant pages in the Web graph, the sooner pages will be indexed and not just crawled. Finally, the user can specify weighting values for tag and title keywords, which is later used for a simple ranking algorithm.

5.2.5 Crawl Results and Index Search

Once the crawl is complete, the program displays a frame containing some information about it along with a text bar to search the indexed pages. As the crawl is launched, a simple stopwatch is also started to record the time it lasts. The result frame furthermore indicates the number of crawled pages and how many out of those were actual tag pages, or more precisely, how many were indexed, since tag pages from which no tags could be extracted are also excluded from this count. The number of gathered keywords is also displayed, sorted by overall, title and tag keywords.

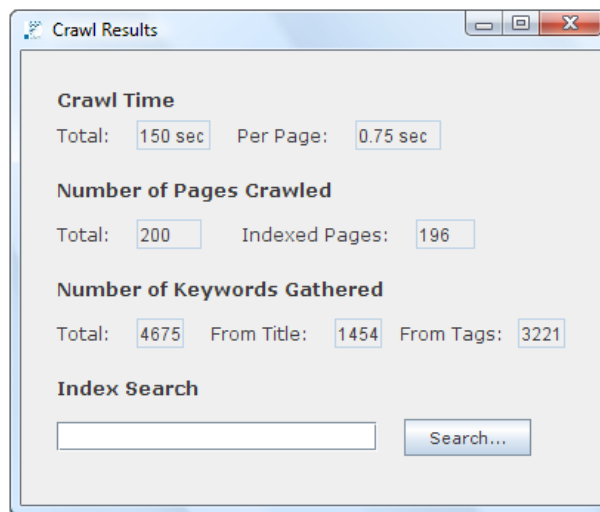


Fig.5.2: Screenshot of the prototype's result frame

Even though not being the prototype's central point of interest, crawling pages would be rather useless if it weren't possible to somehow to consult the set of indexed pages. The program can deal with single or multiple keyword search queries and compare them with the ones extracted from titles and tags. To retrieve matching Web pages, a separate method, *SearchResults()*, is called. The list of all the saved

tags is scanned for the keywords specified by the user and *WebPage* objects stored in another list if they contain at least one matching keyword. All that is then left to do, is to assign these pages a value in order to rank them. This value is simply the sum of the keyword weights, so, if for instance a page has a match in its title keywords and one in its tag keywords, its value will be the sum of the user-defined tag weight and title weight.

Aside from that, before launching the crawl, the user can also chose to save all the indexed pages, that is, their title, URL and tags, into a Excel file stored in the working directory.

5.3 Prototype Evaluation

It is now time to see how well the Web crawler prototype fares according to some general criteria and in particular how well it serves the purpose it was designed to fulfill, namely harvesting user-created metainformation.

5.3.1 Overall Performance

Web crawlers are often evaluated by their ability to retrieve good pages, with “good” defined as being of high relevance to the user, a typically vague metric. To judge the quality of this crawler prototype, it will however not be necessary to define the term of “relevance”, since the focus is a different one. For this reason, it would appear more appropriate to analyze the prototype's performance based on its faculty to harness folksonomy tags.

A first aspect of that is the proficiency of the crawler in identifying tag pages. The objective of being able to extend its reach in terms of additional tag pages is achieved by allowing the user to specify new ones by entering some information that make it possible for the crawler to find the desired metadata. This approach however has its drawbacks. For one, it forces a lot of complexity and effort on the user, who has to indicate the correct URL for a set of tag pages as well as browsing through a page's source code and finding the correct HTML tag. And in addition to knowing exactly what he is meant to do, the user has the full responsibility of the functioning of tag extraction, as there are many possibilities for mistakes and insufficient specifications to sneak in and compromise the latter. This problem is somewhat relieved with a set of already defined tag page specifications the user can choose from. Still, this doesn't guarantee that all the pages at the top of the frontier can be indexed, since the set of pages whose URL begins with the string specified by the user may also contain non-resource pages. Perhaps a more precisely outlined template URL for a set of tag pages would save the crawler some time spent

on pages that can't even be indexed, but this would in turn make adding new tag pages even more complicated.

After establishing in Sect. 4.4 that it would be very difficult to automatically identify tags, a major goal for the Web crawler prototype was to at least be able to do so in a semi-automatic fashion, that is, once it “learned” how to extract tags from some site's pages, it would be able to do that indefinitely. However, the present prototype doesn't store the corresponding information on the disk in order to recover them the next time it is run, a functionality that was left out due to time constraints, which means that they have to be specified anew each time. In the prototype, a specified set of tag pages is also tied to a certain file type, such as photos or videos, in order for the user to indicate before launching the crawl on what kind of resources it has to focus. This is evidently also implemented in the state of mind that several tag page specifications (*ExtractionTemplate* objects in the prototype) may already exist, and that it can be chosen to gather tags only from a restricted set of these.

When it comes to extracting tags, the prototype does rather well, provided the HTML element they are located in is correctly defined. Catching all the text outside of brackets however has the side effect of including unwanted words, like a title introducing the tags, or in the case of Delicious, the tag frequency value. A few stopwords are caught, as well as some user-defined terms, but this doesn't always keep the index from being littered with the occasional non-descriptive keyword. Additionally to that, potentially valid tags can be rendered useless by containing some special characters or spelling errors.

Due to recurring problems in the test phase of the prototype with the memory of 64 MB allocated to the Java Virtual Machine running out of space, the crawl length had to be limited to 200 pages for the final version, although the program occasionally also worked with 250 page crawls, as in the following numerical evaluation.

5.3.2 Numerical Evaluation

Aside from implementational aspects, let's have a look at the prototype's actual results from crawling and indexing some tag pages. In particular, the statistics for crawls on the photo sharing platform Flickr and the video sharing site YouTube¹ will be the center of attention of this numerical evaluation. Crawls with lengths ranging between 50 and 250 pages were run for each set of tag pages. The results can be

¹ Even though YouTube is, as established earlier in this paper, not an actual folksonomy, it is, with its large number of labeled resources, very well suited to demonstrate the prototype's effectiveness in extracting user tags.

seen in Tab. 5.1. All the crawls were started at the corresponding site's home page. The seed page is however not of great importance for the percentage of indexed pages in relation to all the crawled ones, as a platform's resource pages are typically located close to each other in the Web graph. What on the other hand is of relevance for how many tag pages are crawled, is the precision with which their URL can be described. The values in Tab. 5.1 clearly show that on YouTube, apart from a few other pages, probably visited at the start of the crawl, the major part of it is spent on actual video pages, which is due to the fact that their URL can be narrowed down very exclusively. The Flickr crawls, however, extracted tags only on 12 to 19 percent of the visited pages, which stems from the impossibility to determine a URL beginning for this platform that doesn't include non-resource pages, such as for example a particular user's photo gallery, which has the URL format "*http://www.flickr.com/photos/username*" and thus the same address as actual photo pages up to the last slash. Hence, many irrelevant pages are visited without being indexed.

The crawl times per page, although being very consistent for pages from the same site, expose a deep gap in the processing speed between crawls on YouTube and on Flickr. The reason for that is probably that it took much longer to fetch a page from Flickr than from YouTube. As for the extracted keywords, their amount is roughly equal on both platforms with about 17 for each indexed page. These keywords came of course for the most part from the tags and it is obvious that the practice of adding them is in general less wide-spread among YouTube users, with only a little more than two thirds of the keywords

Tag page (Seed page)	Crawl length	Crawl time (per page) in sec	Indexed pages (% of total)	Indexed keywords (per indexed page)	Title keywords (% of total)	Tag keywords (% of total)
YouTube (youtube.com)	50	34 (0.68)	45 (90%)	722 (16)	230 (31.9%)	492 (68.1%)
	100	68 (0.68)	95 (95%)	1380 (14.5)	460 (33.3%)	920 (66.7%)
	150	96 (0.64)	145 (96.7%)	2414 (16.6)	750 (31.1%)	1664 (68.9%)
	200	142 (0.71)	191 (95.5%)	3410 (17.9)	950 (27.9%)	2460 (72.1%)
	250	173 (0.69)	238 (95.2%)	3906 (16.4)	1178 (30.2%)	2728 (69.8%)
Flickr (flickr.com)	50	86 (1.72)	7 (14%)	164 (23.4)	22 (13.3%)	142 (86.7%)
	100	160 (1.6)	15 (15%)	227 (15.1)	32 (14.1%)	195 (85.9%)
	150	242 (1.61)	27 (18%)	459 (17)	70 (15.3%)	389 (84.7%)
	200	339 (1.7)	38 (19%)	641 (16.9)	91 (14.2%)	550 (85.8%)
	250	442 (1.79)	31 (12.4%)	518 (16.7)	72 (13.9%)	446 (86.1%)

Tab.5.1: Results for crawls of different lengths on YouTube and Flickr

found in the folksonomy tags and about 11 of them per indexed page. On Flickr, on the other hand, the keywords from the tags made for around 85 percent of the extracted terms. These numbers may even be somewhat misleading, since, when a keyword is found twice, the one with the lower weight value specified by the prototype user is removed, which in the test crawls was always the keyword from the tags. When those keywords were given a higher weight, it was found that the percentage of tag keywords rose to about 91 percent for Flickr pages and over 80 percent for YouTube pages, which shows that many tag terms were previously evicted because of already being present in the title. Still, it makes more sense to value title keywords higher because of their more concise description of the resource's content and the lower amount of useless information stored in them. Nonetheless, this doesn't undermine the use of harnessing user-created tags to index a page, since they can add a lot of value to an index by addressing a much wider vocabulary than title tags.

Once again, it is worth pointing out that the prototype is merely meant as a demonstration tool for harnessing meta-information that is otherwise not identified and weighted appropriately, a functionality that is supposed to be more of an addition to the traditional current Web crawlers than the center of a standalone application. From this perspective, the present prototype's comparatively slow crawl speed and sometimes low percentage of indexed pages are not alarming, but rather an effect of the program's simplicity and focus on tagged resources. Some possible future extensions and additions are discussed in the next section.

5.3.3 Future Improvements and Extensions

It is apparent that operating the prototype can be confusing to a user unfamiliar with what exact input is required from his side. Before a crawl can be started, numerous elements need to be specified to make it work, such as at least one tag and seed page and some parameters for the crawl. Adding tag pages, in particular, requires a precise knowledge in order to correctly define the URL and HTML tag, in addition to the effort of looking them up in a page's source code. With regard to implementation simplicity, however, the solution used in the prototype was assessed as the optimal one. Since an automatic identification of the folksonomy tags seems impossible due to their lack of standardization, user implication is unavoidable. In a more elaborate application, it could be conceivable to better assist him in the task of pointing the Web crawler in the right direction, for example by having him mark the relevant text on a rendered page instead of just in the source code.

As mentioned before, the actual purpose of harnessing folksonomy tags is to use them for indexing

with a regular Web search engine, and not to have a Web crawler be fully dedicated to finding and extracting them. In such a context, the fact that tag pages cannot be defined precisely enough with their URL not to include other pages would not matter at all, apart from the overhead cost of looking for inexistent HTML tags, since those pages would then simply be indexed based on conventional criteria. This general-purpose crawler would then also need the faculty to store tag page information permanently.

A further useful feature would be to additionally take into account tag frequencies of broad folksonomies in order for the search engine's index to fully benefit from the knowledge created through collaborative tagging. Machine tags can contain valuable and precise meta-information as well. So all in all, Web search engines would have a lot to gain from the advantages provided by folksonomy tags, on one hand because of their reliability in describing a resource's content, especially if they are tied to some value expressing their quality, such as tag frequency, and on the other hand because of the broad vocabulary they can address, a great potential asset in making Web pages accessible to a wide public made up of groups using very different search terms.

6 Conclusion

6.1 Paper Synopsis and Results

The goal of this thesis was to discuss the phenomenon of collaborative tagging and to analyze its advantages and weaknesses as well as its potential use to Web search engines for the indexing of certain resources. To demonstrate how this can be done, a crawler prototype was implemented and has successfully proved that the benefits of folksonomies can easily be harnessed with a simple program, albeit not in a fully automated fashion.

In the first part of this paper, all the relevant theoretical aspects were discussed. In the description of Web search engines, a particular emphasis was given to the Web crawler, and it was found that this module has to face numerous challenges, especially due to its confrontation with a Web growing remarkably fast in size and subject to many changes, both in user behavior and in technology. Thus, crawlers always need to keep up with the latest evolutions on the Web to make sure their corresponding search engine can live up to its users' demands.

Then, the concept of metadata, or metainformation, and some of its forms on the Web, was introduced. Due to its similarity in intention with folksonomies, the Semantic Web, an emerging approach destined to give information a consistent machine-readable meaning through specific standards, is also extensively covered in the theoretical part. While such a design would drastically improve the retrievability of Web pages, the Semantic Web still has a long way to go before becoming the leading standard on the WWW. A more commonly used representation of metadata, and the central subject of this paper, are folksonomies. These unstructured and entirely user-created sets of content-descriptive keywords have many arguments going for them, mainly their simplicity of use and suitability in describing Web documents in a precise way. Their lack of structure obviously also opens the door to many potential flaws and abuses.

In Chap. 4, it is first shown that the purpose of harnessing folksonomies for indexing resides not only in their ability to create metainformation for large sets of resources where otherwise there would be none, but also in potentially capturing the essence of a Web page better than a full-text index would. However, this metainformation is typically not identified by Web crawlers as such. Several possible methods to harness the wisdom of user-created tags were then discussed and the conclusion made, that, due to the lack of a folksonomy standard, it would seem very difficult to have a program recognize

them automatically. The option was also addressed to have the location of the tags be pointed out to it by its user, so that the crawler would be able to autonomously extract the tags when visiting a page containing some.

To show how this design could be put into practical effect, a Web crawler prototype was implemented and described in Chap. 5. It was found that extracting tags can easily be done when their location is specified by the user. Although the prototype was subject to a rather slow performance and did not succeed all of the time in identifying tag pages, the main goal of harnessing folksonomies was clearly achieved.

6.2 Outlook

The use of harnessing folksonomies to index Web pages depends on what the future of collaborative tagging will look like. While folksonomies would undeniably have many benefits to offer to Web search engines, the question is whether they will just remain a side note in the history of the World Wide Web, or find wide-spread use and be applied frequently enough for their being taken into account by Web crawlers to be worthwhile to major commercial search engines. The future evolution of crawlers will always be closely tied to the Web's evolution, with the Semantic Web being the potential next step in improving the retrievability of Web pages with the help of metainformation.

Concerning the prototype developed in the scope of this paper, the next logical step would consist in embedding the elaborated method to gather user-created tags in a full-scale Web crawler, that is, one that would also index all the other pages on its path. The technique of having the program be shown what parts of a Web page should particularly be paid attention to with indexing could obviously also be extended to other cases where a set of pages consistently has relevant information in a specific location.

References

[Alpert & Hajaj 2008]

Alpert, Jesse; Hajaj, Nissan: *We Knew the Web Was Big...*, available: <http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html>, accessed September 5th 2009.

[Bar-Ilan 2003]

Bar-Ilan, Judit: Search Engine Ability to Cope With the Changing Web. In: Levene, Mark; Poulouvassilis, Alexandra (Eds.): *Web Dynamics – Adapting to Change in Content, Size, Topology and Use*. Springer, Berlin, 2003, pp. 195-215.

[Bergman 2001]

Bergman, Michael K.: *The Deep Web: Surfacing Hidden Value*, available: <http://quod.lib.umich.edu/cgi/t/text/text-idx?c=jep;view=text;rgn=main;idno=3336451.0007.104>, accessed September 5th 2009.

[Biglione 2005]

Biglione, Kirk: *Accessible Folksonomies*, available: <http://www.alttags.org/accessibility/accessible-folksonomies/>, accessed October 19th 2009.

[Borodin et al. 2005]

Borodin, Allan; Roberts, Gareth O.; Rosenthal, Jeffrey S.; Panayiotis, Tsaparas: *Link Analysis Ranking: Algorithms, Theory, and Experiments*, available: <http://www.cs.helsinki.fi/u/tsaparas/publications/toit.pdf>, accessed October 13th 2009.

[Brin & Page 1998]

Brin, Sergey; Page, Larry: *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, available: <http://infolab.stanford.edu/pub/papers/google.pdf>, accessed October 13th 2009.

[Broder 2002]

Broder, Andrei: *A Taxonomy of Web Search*, available: <http://www.sigir.org/forum/F2002/broder.pdf>, accessed September 10th 2009.

[Chakrabarti et al. 1999]

Chakrabarti, Soumen; van den Berg, Martin; Dom, Byron: *Focused Crawling: A New Approach to Topic-Specific Resource Discovery*, available: <http://mainline.brynmawr.edu/Courses/cs380/fall2006/prelim.pdf>, accessed October 13th 2009.

[Cho & Garcia-Molina 2003]

Cho, Junghoo; Garcia-Molina, Hector: *Effective Page Refresh Policies for Web Crawlers*, available: <http://rose.cs.ucla.edu/~cho/papers/cho-tods03.pdf>, accessed October 13th 2009.

[de Kunder 2009]

de Kunder, Maurice: *The Size of the World Wide Web*, available: <http://www.worldwidewebsite.com/>, accessed September 5th 2009.

[Delicious 2009]

Delicious: *Explore Tags on Delicious*, available: <http://delicious.com/tag/>, accessed October 20th 2009.

[Delicious 2009 (2)]

Delicious: *Everyone's bookmarks for "Technology Review: The Authority on the Future of Technology" on Delicious*, available: <http://delicious.com/url/35092bab9fda1203898ad2267864d263>, accessed November 5th 2009.

[Dictionary.com 2009]

Dictionary.com: "Ontology", available: <http://dictionary.reference.com/browse/ontology>, accessed October 13th 2009.

[Dolog & Nejd1 2007]

Dolog, Peter; Nejd1, Wolfgang: Semantic Web Technologies for the Adaptive Web. In: Brusilovsky, Peter; Kobsa, Alfred; Nejd1, Wolfgang (Eds.): *The Adaptive Web – Methods and Strategies of Web Personalization*. Springer, Berlin, 2007, pp. 697-719.

[DomainTools 2009]

DomainTools: *Domain Counts & Internet Statistics*, available: <http://www.domaintools.com/internet-statistics/>, accessed September 5th 2009.

[Flickr 2009]

Flickr: *Popular Tags on Flickr*, available: <http://www.flickr.com/photos/tags/>, accessed October 20th 2009.

[Flickr 2009 (2)]

Flickr: *power curve on Flickr*, available: <http://www.flickr.com/photos/43845297@N06/4033717923/>, accessed October 20th 2009.

[Fusco 2007]

Fusco, P.J.: *Tags, Customized Search, and SEO*, available: <http://www.clickz.com/3627385>, accessed October 13th 2009.

[Griffiths 2005]

Griffiths, Richard T.: *Search Engines*, available: <http://www.leidenuniv.nl/letteren/internethistory/index.php3-c=7.htm>, accessed October 13th 2009.

[Gruber 1996]

Gruber, Thomas, R.: *Toward Principles for the Design of Ontologies Used for Knowledge Sharing*, available: http://www.itee.uq.edu.au/~infs3101/_Readings/OntoEng.pdf, accessed October 13th 2009.

[Ho 2009]

Ho, Matthew: *The Wisdom of Crowds (With Bookmarks)*, available: <http://inspiredworlds.com/2009/01/06/the-wisdom-of-crowds-with-bookmarks> accessed October 13th 2009.

[Hotho et al. 2006]

Hotho, Andreas; Jäschke, Robert; Schmitz, Christoph; Stumme, Gerd: *Information Retrieval in Folksonomies: Search and Ranking*, available: http://www.kde.cs.uni-kassel.de/hotho/pub/2006/seach2006hotho_eswc.pdf, accessed October 13th 2009.

[Kashyap & Sheth 1997]

Kashyap, Vipul; Sheth, Amil: *Semantic Heterogeneity in Global Information Systems the Role of Metadata Context and Ontologies*, available: <http://dit.unitn.it/~p2p/RelatedWork/Matching/KS97.pdf>, accessed October 13th 2009.

[Kleinberg 1999]

Kleinberg, Jon M.: *Authoritative Sources in a Hyperlinked Environment*, available: <http://scicomp.pitt.edu/~milos/courses/cs3750/Readings/Link-analysis/kleinberg99authoritative.pdf>, accessed October 13th 2009.

[Lassila & Swick 1999]

Lassila, Ora; Swick, Ralph R.: *Resource Description Framework (RDF) Model and Syntax Specification*, available: <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222>, accessed October 13th 2009.

[Li 2002]

Li, Wei: *The First Search Engine, Archie*, available: <http://www.isrl.illinois.edu/~chip/projects/timeline/1990archie.htm>, accessed October 13th 2009.

[Liu 2007]

Liu, Bing: *Web Data Mining – Exploring Hyperlinks, Contents, and Usage Data*, Springer, Berlin, 2007.

[Maedche & Staab 2001]

Maedche, Alexander; Staab, Steffen: *Ontology Learning for the Semantic Web*, available: http://www.aifb.uni-karlsruhe.de/WBS/Publ/2001/OLfSW_amasst_2001.pdf, accessed September 10th 2009.

[Markov & Larose 2007]

Markov, Zdravko; Larose, Daniel T.: *Data Mining the Web – Uncovering Patterns in Web Content, Structure, and Usage*, John Wiley & Sons, New York, 2007.

[Mathes 2004]

Mathes, Adam: *Folksonomies - Cooperative Classification and Communication Through Shared Metadata*, available: <http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html>, accessed October 13th 2009.

[Merholz 2004]

Merholz, Peter: *Metadata for the Masses*, available: <http://www.adaptivepath.com/ideas/essays/archives/000361.php>, accessed October 13th 2009.

[Micarelli & Gasparetti 2007]

Micarelli, Alessandro; Gasparetti, Fabio: Adaptive Focused Crawling. In: Brusilovsky, Peter; Kobsa, Alfred; Nejdl, Wolfgang (Eds.): *The Adaptive Web – Methods and Strategies of Web Personalization*. Springer, Berlin, 2007, pp. 231-262.

[Ng et al. 2001]

Ng, Andrew Y.; Zheng, Alice X.; Jordan, Michael I.: *Stable Algorithms for Link Analysis*, available: <http://ai.stanford.edu/~ang/papers/sigir01-stablelinkanalysis.pdf>, accessed October 13th 2009.

[Ntoulas et al. 2004]

Ntoulas, Alexandros; Cho, Junghoo; Olston, Christopher: *What's New on the Web? The Evolution of the Web From a Search Engine Perspective*, available: <http://www.cs.brown.edu/courses/cs253/papers/www04-ntoulas.pdf>, accessed October 13th 2009.

[Obitko 2007]

Obitko, Marek: *Ontologies and Semantic Web*, available: <http://obitko.com/tutorials/ontologies-semantic-web/?/stor1/obitko>, accessed October 13th 2009.

[Palmer 2001]

Palmer, Sean B.: *The Semantic Web: An Introduction*, available: <http://infomesh.net/2001/swintro>, accessed October 13th 2009.

[Pant et al. 2003]

Pant, Gautam; Srinivasan, Padmini; Menczer, Filippo: Crawling the Web. In: Levene, Mark; Poulouvasilis, Alexandra (Eds.): *Web Dynamics – Adapting to Change in Content, Size, Topology and Use*. Springer, Berlin, 2003, pp.153-177.

[Pink 2005]

Pink, Daniel H.: *Folksonomy*, available: http://www.nytimes.com/2005/12/11/magazine/11ideas1-21.html?_r=2, accessed October 13th 2009.

[Shadbolt 2006]

Shadbolt, Nigel; Hall, Wendy; Berners-Lee, Tim: *The Semantic Web Revisited*, available: http://eprints.ecs.soton.ac.uk/12614/1/Semantic_Web_Revisted.pdf, accessed October 13th 2009.

[Smith et al. 2004]

Smith, Michael K.; Welty, Chris; McGuinness, Deborah L.: *OWL Web Ontology Language Guide*, available: <http://www.w3.org/TR/owl-guide/>, accessed October 13th 2009.

[Stock 2007]

Stock, Wolfgang G.: *Folksonomies and science communication*, available: http://www.walt.phil-fak.uni-duesseldorf.de/infowiss/admin/public_dateien/files/1/1194272247inf_servic.pdf, accessed October 13th 2009.

[Stuckenschmidt 2004]

Stuckenschmidt, Heiner; van Harmelen, Frank: *Information Sharing on the Semantic Web*, Springer, Berlin, 2004.

[Sullivan 2002]

Sullivan, Danny: *Death of a Meta Tag*, available: <http://searchenginewatch.com/2165061>, accessed October 13th 2009.

[Vander Wal 2005]

Vander Wal, Thomas: *Explaining and Showing Broad and Narrow Folksonomies*, available: http://www.personalinfocloud.com/2005/02/explaining_and_.html, accessed October 13th 2009.

[Vander Wal 2007]

Vander Wal, Thomas: *Folksonomy Coinage and Definition*, available: <http://vanderwal.net/folksonomy.html>, accessed October 13th 2009.

[Visser 2004]

Visser, Ubbo: *Intelligent Information Integration for the Semantic Web*, Springer, Berlin, 2004.