Bulletin of the Technical Committee on

Data Engineering

June 2009 Vol. 32 No. 2

IEEE Computer Society

Letters

Special Issue on New Avenues in Search

Conference and Journal Notices

The TC on Data Engineering
Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TC on Data Engineering web page is
`http://tab.computer.org/tcde/index.html`.

The Data Engineering Bulletin
The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at
`http://tab.computer.org/tcde/bull_about.html`.

Letter from the Editor-in-Chief

The Current Issue

The area of ”web search” is surely an area of great commercial interest. One can follow this commercial story via the newspapers or search for it on the web– the ongoing saga of Microsoft courting and being rebuffed by Yahoo ..., Google market share numbers continuing to increase to the point of bringing DOJ interest, Microsoft re-introducing its search effort, now called ”Bing!”, new international entrants to the search market, etc. This blizzard of news tends to blur the fact that ”search” is also a technical area where ”new avenues” are being pursued.

It is these new avenues in search that is the subject of the current issue. What is terrific about this area is its combination of commercial importance, hard technical problems, and continuing insights. We are no longer satisfied with answering the straightforward key word search over html pages. One effort is to discover more fully what the user intent is. Another is to search the web more deeply, in many cases for underlying, and partially hidden structured information. Collaborative search is yet another direction. This area is rife with both challenges and opportunities.

Sihem Amer-Yahia is the editor for this special issue. Her letter provides a more complete overview of the current issue. Being at Yahoo, Sihem is very much aware of the work going on in the search area, who is doing it, and what is important about it. She has put that awareness to good use in assembling the current issue. This issue provides a way both to introduce yourself to the search area and also to learn some of the latest thinking from researchers active in the area. I want to thank Sihem for doing a fine job on this issue, which I expect will be read with great interest not only by our technical community but by the more business oriented folks as well.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editor

Search has become a daily activity. Due to its popularity, most of us expect to be able to use search everywhere and, when we do, want it to work, that is, return good results regardless of the complex nature of the data or the task we are looking to achieve. Despite its popularity, search is yet to be improved. This issue contains several messages about how we should go about making search better. The nature of the data and its richness, structured objects, XML data, user-contributed content, require various semantic interpretations of search. The user intention, whether searching personal information, looking to explore topics or finding product information, is another dimension affecting the effectiveness of search results. Finally, the advent of the social Web is changing users' expectations and their perceptions of data quality, and hence, the requirements of search.

The first article, by Kumar and Tomkins, gives us insights on how people search based on a large-scale analysis of search and toolbar logs. It observes, among other things, that more than half of search queries are about structured objects. The authors then study the relationship between search and e-commerce queries as they represent a monetizable user intent.

The second article, by Anand Rajaraman, describes Kosmix, a general-purpose topic exploration engine which goes beyond Web search to extract Deep Web information using taxonomies.

The following two articles, by Marian and Wang, and, Dittrich, Vas Sallez and Blunschi, discuss the state of the art in searching personal information. The first article describes a method which uses query approximation to search personal content across file boundaries. The second article describes the interplay between search and data integration. Users can not only search their data collections but also semantically integrate it as they search.

The fourth article reports on experiences and challenges in building an XML search engine. one main distinction of searching XML data is the ability to determine which portion of a document is most relevant to a user's request, as opposed to traditional Web search, where entire documents are returned.

Searching user-contributed and shared content is tackled in the last two articles by Abiteboul and Polyzotis and, Agichtein, Gabrilovich and Zha. The first article describes data rings, an infrastructure, which enables users to declaratively share and consume other users' content. The second article applies machine learning techniques to finding appropriate ranking functions when searching social content.

I would like to thank the authors who graciously volunteered their time and effort in putting together this special issue.

Happy reading!

Sihem Amer-Yahia
Yahoo! Labs
New York, NY

A Characterization of Online Search Behavior

Ravi Kumar Andrew Tomkins
Yahoo! Research
701 First Ave.
Sunnyvale, CA 94089, USA.
{ravikumar,atomkins}@yahoo-inc.com

Abstract

In this paper we undertake a large-scale study of online user search behavior based on search and toolbar logs. We identify three types of search: web, multimedia, and item. Together, we show that these different flavors represent almost 10% of all online pageviews, and indirectly result in over 21% of all pageviews.

We study search queries themselves, and show that more than half of them contain direct references to some type of structured object; we characterize the types of objects that occur in these queries. We then revisit the relationship search and navigation specifically in the context of e-commerce, and consider how search aids users in online shopping tasks.

1 Introduction

The online environment has shifted dramatically in the last fifteen years, with orders of magnitude growth in both users and content, as well as significant expansion of the capabilities users expect. Every day, new websites emerge seeking to transform online paradigms and woo users with new types of offerings. Social networking sites are skyrocketing in popularity and levels of user engagement, while traditional online communications paradigms like email continue to see significant usage. Search over pages, listings, and multimedia is increasing in usage and well as sophistication of result sets. And increasingly complex user tasks, from purchasing replacement laptop batteries to booking family vacations in foreign lands, are migrating from offline to online venues. We do not have a complete understanding of these dynamics, partly because the rate of change is high, and partly because there no publicly available data sources that offer a large-scale picture of cross-web user behavior.

In this paper we undertake such a study based on data collected through the Yahoo! toolbar. We analyze a large sample of over fifty million user pageviews collected over an eight-day period in March of 2009 from users who have installed the Yahoo! toolbar and agreed to collection of their data for purposes including this type of analysis. In addition, we augment this dataset in certain areas with data taken from Yahoo! search logs and editorially annotated in various ways.

Employing toolbar data as our primary form of data collection introduces some selection bias on the users with installed toolbars, and captures only web pageviews. Mobile usage is not included in our study, nor is use of AJAX for asynchronous fetching. Nonetheless, we provide a characterization of pageviews, which remains the dominant mechanism by which users interact with the web.

Our goal in this study is to characterize search behavior, which we interpret broadly. We do not restrict our attention to web search, but we consider also multimedia search as well as search of databases of items, as provided by the search boxes of eBay or Amazon. We also consider user behavior after leaving the search results page. Main web search represents 6.2% of all pageviews, multimedia search 1.4%, and item search another 1.4%. These 9% of search pageviews are responsible for an additional 8.9% of pageviews that are direct referrals, and yet another 3.5% of indirect referrals, resulting in a total of 21.4% of overall pageviews that are either on search, or reached through search.

Thus, search is "responsible" for roughly one in five pageviews online, so it is of great interest to understand exactly what these search queries cover. There have been a wide range of studies of search query types starting with the seminal work of Broder [3]. We choose to explore queries through the lens of structured data, with an eye to understanding the user appetite for structured information on the web. We will show that more than half of all queries contain a direct reference to a structured object that is central to the query, in a sense to be made formal below. We also present a taxonomy of types of structured information, and give statistics of their occurrence frequencies in search logs. We suggest that the future of search will include increasing focus on gathering, understanding, and when appropriate serving structured information from these domains.

Based on the prevalence of product queries of different types, we explore this claim further through a small study of the connection between search and online commerce. We automatically identify the action of adding an item to a shopping cart, and then study how users arrive at these checkout pageviews. We show that 20% of checkout pageviews are reached through search. We also characterize behavior with respect to time spent before performing the final search of the session, then time spent between the search and entry to the e-commerce site, and finally time spent on the e-commerce site before reaching the checkout pageview. Pre-search, post-search, and on-site periods respectively represent on average about 1/2, 1/6, and 1/3 of total pageviews in the path.

The remainder of the paper is organized as follows. Section 2 covers related work. Then in Section 3 we present our taxonomy of online search, and study the connection between search and navigation. Section 4 presents a study of structured data on the web. Section 5 then studies specifically the relationship between search and e-commerce. Finally, in Section 6 we present our concluding thoughts.

2 Related work

There have been several approaches to model user interaction with search engines. For example, Lau and Horvitz [18] introduce a probabilistic Bayesian network in order to predict query-query topic transitions, by examining the context of the query along with inter-query time period. Radlinski and Joachims [23] identify sequences of queries on the same topic using features based on shared words in the queries; see Spink et al [24] for the problem of topic switching and multi-tasking in query sessions. Downey, Dumais, and Horvitz [9] introduce an expressive language for describing searching and browsing behavior on the client side. The language is based on a state-machine representation and provides a unified framework for analyzing general models of user behavior, including many server-side models that were studied earlier [7, 18, 15, 17, 23]. They also construct machine-learned models to predict the next action of the user using features derived from their behavior thus far. Very recently, Guo et al [11] and Chappelle and Zhang [6] propose Bayesian click models to infer relevance of search results.

Web browsing activity beyond search has also been extensively studied and modeled, e.g., [5, 19]. Bucklin and Sismeiro [4] use a server-side log to model the browsing behavior of visitors to a website; they focus on "stickiness" of the website and analyze repeat visit patterns. Park and Fader [22] develop a stochastic timing model of cross-site user visit behavior in order to use information from one site to explain the behavior at another. Johnson et al [14] study online search and browsing behavior across competing e-commerce sites.

Recently, Dumais et al [10] study browsing behavior after the user departs the search engine and begins to follow an information thread through the web.

Search queries have been analyzed from a variety of viewpoints, including data mining query logs and analyzing user activities from query sessions. There is an extensive literature on mining query logs and user search activities for various search-related applications including query suggestions [8, 17], query expansion [7], and ranking [1]. We refer to the survey by Baeza-Yates [2] for further details. Identifying sessions and session boundaries in query streams has been a hot research topic, e.g., [13]. This is typically done by using an off-the-shelf classifier and developing features based on query terms and user activities, including dwell times and timeouts; see [12, 21, 20]. Jones and Klinkner [16] develop effective classifiers that go beyond timeout-based features in order to identify sessions and goals, and more fine-grained task and subtask boundaries.

3 Search and navigation

In this section we study the prevalence of different types of searches, and their impact on web navigation. We first introduce the toolbar dataset used to derive most of the results in this paper. The toolbar dataset contains a random sample of US users drawn from Yahoo! toolbar logs over a one week period from March 18, 2009 to March 24, 2009. The dataset has been scrubbed to remove information that may allow identification of any user in the set, and to remove any other sensitive material. The number of pageviews in the dataset is around 50M, with around 7.2M pageviews per day over the weekdays and around 6.5M per day over the weekends. Certain of our results also employ auxiliary datasets generated from Yahoo! search logs; we describe these datasets as we introduce them.

3.1 Search result pageviews

Pages containing search results may be broken into three key types: (1) *main search* represents pageviews on web search sites, restricted to pageviews that are part of web search, rather than other types of search, (2) *multimedia search* captures search for videos, images, music, or other forms of multimedia, and (3) *item search* captures search through a database of listings, as provided by the search boxes of Amazon, eBay, or Craigslist.

We selected a random sample of 1000 pageviews from the toolbar dataset, and manually classified each pageview into one of the three types of search pageviews above, or "Other." Search result pages overall represent 9% of total pageviews, larger than we anticipated. Search is dominated by main search (standard web search) at 6.2% of total pageviews, while multimedia and item search each represent 1.4% of pageviews.

Automatically recognizing search pageviews. To perform a larger-scale analysis of these pageviews, we develop simple automated recognizers for different types of searches based on hand-built rules. For main search, we construct recognizers for the five largest US search engines (as our toolbar dataset contains US pageviews): Yahoo, Google, MSN, Ask, and AOL. For multimedia search, we construct recognizers for multimedia search result pages from the above search engines, along with Youtube, Hulu, Flickr, and Picasa. For item search, we construct recognizers for the URLs corresponding to search results from the most popular providers of listings in the dataset: Amazon, eBay, Craigslist, Imdb, Singlesnet, Careerbuilder, and Leboncoin. Additionally, we employ a set of general rules matching any URL that has a search-like parameter embedded in the URL, such as `&q=madonna` or `&search=madonna`.

Using the automated recognizers, we observe that 5.1% of pageviews are classified as main search, 1.5% as multimedia search, 0.5% as item search, and 1.7% as other searches. Observe first that these recognizers overall capture 8.8% of pageviews, and our earlier manual classification study showed that the correct fraction is 9% of pageviews ($\pm$1.8% at 95% confidence). Thus, our recognizers possibly capture almost all search pageviews. Second, note that the fractions of pages we capture for main search and multimedia search are quite similar to the fractions of the overall population as determined by our manual study. Item listings, however, have a longer tail, and thus we capture a smaller fraction of these in our explicit recognizer.

3.2 Referrals

To begin, we explore the interactions between search and web navigation by considering the source of the referrer link that took a user to a particular page. 34.4% of pageviews have no such referrer, because the user arrived at the page by selecting a bookmark or by directly typing into the browser's navigation bar or by clicking on links from other applications, and so forth. We call these pageviews *starting points,* and all other pageviews *referrals.* 5.3% of all pageviews have a referral from main search, 1.4% from multimedia search, 0.6% from item search, and 1.5% from other search. Thus, each category of search pageviews tends to result in around one follow-on click on average.

Of course, that follow-on click may result in yet more downstream clicks. Consider for example a user who visits Yahoo!, performs a search, explores several search results, discovers a high-quality hub page, and continues to browse from the hub page. In this scenario, the object of study should be the *referral chain.* As a user can potentially follow multiple links from a single page (either through multi-tab browsing, or through use of the back button), the chain is in fact a tree. And as there can be multiple starting points within a session, the tree might become a forest. We refer to this artifact as the *referral forest* of a session.

In the context of a referral forest, we ask the following question: given that a user has arrived at a certain page, was search involved in getting the user there? More specifically, is there a search page in the path from the current page to the root of its tree in the referral forest?

16.2% of all pageviews have a main search page as an ancestor. This value is 3.3% for multimedia search, 0.9% for item search, and 4.4% for other search. In aggregate, 21.4% of pageviews have some type of search result page as an ancestor. Thus, about one out of six pageviews are reached directly or indirectly through web search, and more than one out of five pageviews are reached directly or indirectly through some kind of search. These calculations count the pageview for the search result itself, but do not count the pageview to navigate to the search engine.

Search session statistics. Given that search is in some way responsible for a significant fraction of pageviews, we present some basic statistics of a *search session*, defined as follows. We say that a pageview is a *search root* if it is labeled as main search, and does not have any ancestor in the referral forest labeled as main search. A search session for a particular search root is then taken to be the set of pageviews in the subtree rooted at the given search root. Notice that, by definition, search sessions may overlap with one another in time, but each pageview belongs to at most one search session. The mean number of search sessions per user per day is 0.57. The average number of pageviews in a search session is 6.3, and the average depth of the subtree rooted at a search root is 4.2. This implies that users do not simply perform a search and then explore the immediate results; they typically explore longer paths from the initial search results page.

In aggregate, then, a search session takes around 6 pageviews. Of these, one is the initial search, on average there is one additional search, and the remaining four pageviews are off-search. Figure 1 gives a more detailed breakout of this picture. It shows for each session length what fraction of pageviews is spent on the search engine, versus browsing on the rest of the web. As expected, longer sessions correspond strongly to more time spent engaging off the search engine with content on the rest of the web.

Assigning credit for pageviews. We have seen that search is a contributor to a significant fraction of pageviews (around one in five). However, any particular pageview may be reached by a complex chain of referrals of which search is only a piece. A different perspective may be gained by attempting to allocate credit for the pageview across the ancestors of the pageview. We consider the following two measures for assigning credit among the ancestors of a pageview. (1) *Root credit:* for each pageview, assign one unit of credit to the root of the referral

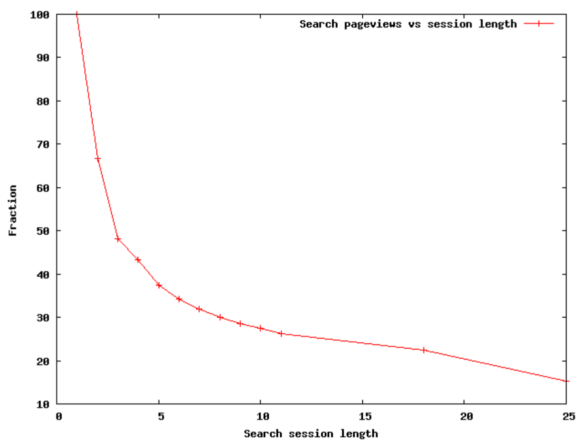


Figure 1: Fraction of search session pageviews spent at search engine, as a function of the total pageviews in session.

tree in which the pageview appears. (2) *Amortized credit:* for each pageview, assign one unit credit, but spread this unit of credit across all ancestors of the pageview, with the root getting a small amount of credit, the next node getting twice as much, and so forth until the pageview itself is reached. Under the root credit measure, main search gets credit for 7.7% of pageviews, multimedia search for 0.5%, item search for 0.1%, and other search for 1.1%. For amortized credit, the corresponding numbers are 5.9%, 1.7%, 0.4%, and 1.9%.

Thus, the various approaches to assigning credit present a consistent picture: the fraction of search pageviews of each type are roughly akin to the fraction of direct referrals from search pageviews of each type, which are themselves roughly akin to the fraction of indirect referrals using both root and amortized methods of credit allocation. This suggests that, while there is likely to be a search pageview on the path to interesting content, such pageviews are a stepping stone along the way, rather than the sole attributable provider of access to key content.

4 Structured data in search

In this section we study the extent to which search queries include references to *structured objects*, or simply *SO*s. We will address this question using a dataset of search queries collected from the Yahoo! web search engine log in the third quarter of 2008. To form this dataset, queries were sampled randomly from the log, so the sample contains the appropriate fraction of head versus tail queries.

Examples of SOs include restaurants, products, cars, real-estate listings, cities, bands, sports teams, celebrities, sports players, and companies. To be considered an SO, a candidate should have non-trivial metadata available on the public web in a format that could be extracted for automated processing. Thus, it is possible that an entity might not be an SO today, but as more detailed metadata becomes available, it might become an SO per this definition at a later date.

One may draw a distinction between categories such as high-definition TVs versus instances such as the Samsung DLP8187, versus listings such as a particular Samsung DLP8187 being sold on eBay. The analysis below treats all three of these as SOs, as the intention is to provide some quantitative insights into the types of user needs that can be fulfilled more effectively based on repositories of SOs objects and technology for interpreting content through the lens of a universe of SOs. Understanding the particulars of listings versus instances or categories is an interesting area for future work.

We place each query into one of the following categories: (1) *Central SO*. The query directly references an SO, and furthermore, the SO is central to the context, rather than just a peripheral mention. For instance, in the query `south florida style long skirts`, there is a reference to the SO Florida, but the reference is peripheral; the primary object of interest to the user is a particular type of skirt. Likewise, in the query `disney coloring pages`, the reference to the SO Disney is peripheral. (2) *Peripheral SO.* The query directly references an SO, but the reference is peripheral to the central meaning of the query. (3) *Topic/Concept.* The query does not contain an SO, but contains a topic or concept of the type that might appear in Wikipedia. In some cases the distinction between a topic/concept versus an SO is subtle. Some examples of topics/concepts are Justice, Socialism, Hunger, I-80, the Vancouver Sky Train, the NFL Draft, and the Nuclear Test Ban Treaty. In time, some of these might become SOs as more and more metadata becomes available online. (4) *Other.* The query contains neither an SO nor a Topic/Concept.

In our search dataset, central SOs account for 52.9% of the queries, peripheral SOs account for 4.7%, and Topic/Concept account for 8.5%. Of the remaining 33.8% (i.e., other), about 10% are URL queries, and another roughly 12% are non-URL navigational queries. Thus, of queries that do not contain an SO or a Topic/Concept, about 2/3 are navigational in nature. The remainder are various types of informational and transactional queries. The most striking aspect of this part of the study is that an actual majority of search queries contain central references to an SO, suggesting that an increasingly sophisticated understanding of structured data is key to meeting the needs of search users.

For queries that are labeled SO central, we also break out a few sub-categories as follows: (1) *Event.* Generally something that takes place at a point in space and time, although we also consider recurring events and distributed events to belong to this category. Examples include concerts, conventions, elections, movie releases, and so forth. (2) *Games.* Board games, computer games, multi-player online games, etc. (3) *Notable person.* The SO is a notable person, such as a celebrity, an actor or actress, a sports figure, a well-known blogger, a politician or world leader, a prominent businessperson or scientist, and so forth. (4) *Ordinary person.* The SO is a person, but not one of broad interest. An ordinary person is expected to be of interest primarily to people who are personally acquainted with him or her. (5) *Specific product* and *general product.* The SO is a product, either referring to it specifically (e.g., Samsung DLP8187) or generically (e.g., HDTV). (6) *Places.* The SO is a geographical location, at any level of granularity. (7) *Business Cat/Service.* The SO refers to businesses or services (e.g., laundromat). (8) *Health.* SOs related to medical issues, such as conditions, symptoms, drugs, and treatments. (9) *Real estate.* SOs related to real estate, including apartments and rentals. (10) *Media title.* This SO refers to CD albums, movies, and other media releases. (11) *Organization.* The SO refers to any organization such as a company, non-profit organization, local or national government.

In measuring occurrences of these categories, we may encounter queries that contain multiple SOs. In this case, we assign weight $1/k$ to each of k SOs occurring in the query. Figure 2 shows the breakdown of SOs in terms these categories. There are four categories that occur in more than 10% of queries: Organization is the dominant category (occurring in fully 1/3 of queries) followed by notable person, specific product, and media title. Not coincidentally, these are areas in which search engines already invest significant effort to develop top-of-page informational boxes that present appropriate structured information allowing users to either fulfill their goal without leaving the page, or to engage with the object directly in order to pursue a particular need.

5 Connections between search and e-commerce

We have focused above on the interactions between search and navigation, and on the nature of search queries. General and specific product queries represent around 20% of overall search queries, and are of great interest as they represent a monetizable user intent. In this section we dig a little more deeply into the impact of search on online purchasing, using the toolbar dataset described earlier.

Due to concerns around sensitive information, we do not consider URLs that represent purchase behavior, but instead, we recognize the activity of adding an item to an online shopping cart. To perform this task, we construct a simple recognizer that identifies terms such as `shopping cart`, `checkout`, etc., that are present

Category	% queries
Event	2.31
Games	1.15
Notable person	13.08
Ordinary person	4.42
Specific product	11.35
General product	8.56
Places	4.90
Business Cat/Service	7.69
Health	1.35
Real estate	1.15
Media title	10.10
Organization	33.94

Figure 2: Breakdown of SO central into object categories.

in the directory path/cgi-parameter of the pageview URL; we call these the *checkout pageviews*. Note that this is only an approximation since there is no sure way to ascertain if the transaction actually happened.

We will consider the role that search plays in reaching a checkout pageview. This analysis is necessarily incomplete—for instance, we do not capture the significant impact search may have on so-called ROBO purchase (research online, buy offline), in which users read reviews, compare models, check prices online, but consummate the purchase in person at a local store.

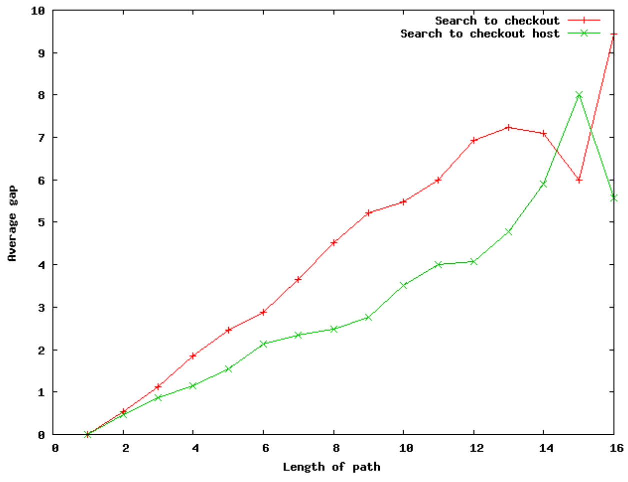
Search contributions to checkout pageviews. We consider each checkout pageview, and follow the referral chain backward to identify the first occurrence of a search pageview of any type. 20.1% of all checkout pageviews have a search ancestor, so about one in five checkout pageviews are reached directly or indirectly through search. 11.2% of checkout pageviews in fact have a search pageview as the root of the path.

Of the roughly 20% of checkout pageviews that have a search ancestor, 36% of these result from a search on the same host as the checkout, and the remainder represent an offsite search in which the user arrives at the e-commerce site through use of an external search engine.

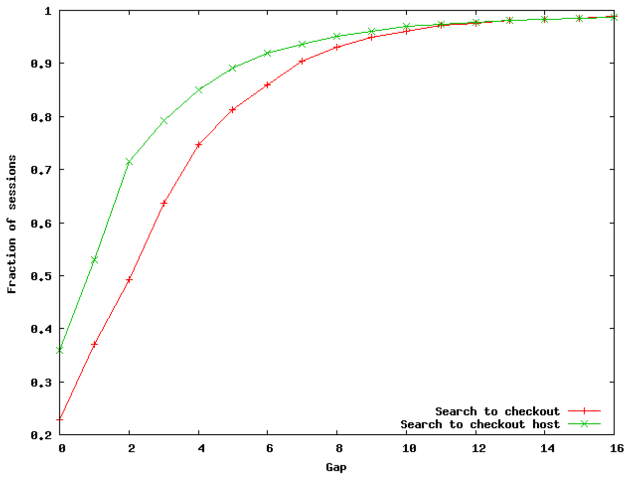
Properties of paths to checkout pageviews. For checkout pages with an offsite search ancestor, we now consider the number of hops to get from the search page to the checkout page. The top, red curve of Figure 3(a) shows these values as a function of the overall path length from root to checkout pageview. The curve grows roughly linear, but with slope slightly above 1/2, indicating that on average a user encounters a search page slightly more than halfway through a session that culminates in a checkout pageview.

The lower, green curve in Figure 3(a) shows the gap from the search pageview to the first pageview on the host at which the checkout pageview occurs. The pattern is similar, but the slope is closer to 1/3. The gap between these curves represents the time spent on the host. The picture emerging is that conditional on performing an offsite search and then later a checkout pageview, the user typically spends approximately 2/3 of the intervening hops getting to the site, and the remaining 1/3 of hops getting to the page hosting the checkout pageview. These values are all based on averages, so a more detailed perspective would emerge from a study of the individual sessions.

Equipped with this view of behavior as a function of pathlength, we now turn in Figure 3(b) to the cumulative distribution of the gaps themselves. The lower, red curve shows the distribution of number of hops from search to the checkout pageview. The upper, green curve shows the distribution of number of hops from search to the host upon which the checkout pageview occurs. In 85% of sessions, the user reaches the host in four hops or fewer, and in 75% of sessions, the user in four hops or fewer reaches the checkout pageview itself.



(a) Average gaps as a function of the path length.



(b) Distribution of gaps.

Figure 3: Analysis of search to buy behavior.

6 Conclusions

We have shown that the search paradigm drives significant direct and indirect traffic. Roughly 90% of search queries are either navigational, or contain reference to some structured object, topic, or concept. These are dominated by queries that contain a direct central reference to an object. Our exploration of the paths taken by users from search to checkout pageviews, in agreement with other work (e.g., [10]), shows that a significant amount of effort ensues between the search query and the fulfillment. We conclude that the impact of search on online activity is more significant than we had anticipated on undertaking this work, and that the impact of structured data on search, while already large, leaves significant headroom for future extensions of the search product.

Acknowledgments

We gratefully acknowledge ConnieAlice Hungate, Amy Holzer, Anneliese Eisentraut, Sarah Ellinger, and Teresa Block for providing the data used in Section 4.

References

[1] E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *Proc. of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 19–26, 2006.

[2] R. Baeza-Yates. Applications of web query mining. In *Proc. of the 27th European Conference on Information Retrieval*, pages 7–22, 2005.

[3] A. Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, 2002.

[4] R. E. Bucklin and C. Sismeiro. A model of web site browsing behavior estimated on clickstream data. *Journal of Marketing Research*, 11:249–267, 2003.

[5] L. D. Catledge and J. E. Pitkow. Characterizing browsing strategies in the world-wide web. *Computer Networks and ISDN Systems*, 27(6):1065–1073, 1995.

[6] O. Chappelle and Y. Zhang. A dynamic Bayesian network click model for web search ranking. In *Proc. of the 18th International Conference on World Wide Web*, pages 1–10, 2009.

[7] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *Proc. of the 11th International Conference on World Wide Web*, pages 325–332, 2002.

[8] H. Daume and E. Brill. Web search intent induction via automatic query reformulation. In *Proc. of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies*, pages 49–52, 2004.

[9] D. Downey, S. Dumais, and E. Horvitz. Models of searching and browsing: Languages, studies, and applications. *Journal of the American Society for Information Science and Technology*, 58(6):862–871, 2007.

[10] D. Downey, S. Dumais, D. Liebling, and E. Horvitz. Understanding the relationship between searchers' queries and information goals. In *Proc. of the 17th ACM Conference on Information and Knowledge Management*, pages 449–458, 2008.

[11] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *Proc. of the 18th International Conference on World Wide Web*, pages 11–20, 2009.

[12] D. He, A. Goker, and D. J. Harper. Combining evidence for automatic web session identification. *Information Processing and Management*, 38:727–742, 2002.

[13] B. J. Jansen, A. Spink, and T. Saracevic. Real life, real users, and real needs: A study and analysis of user queries on the web. *Information Processing and Management*, 36:207–227, 2000.

[14] E. J. Johnson, W. M. Moe, P. S. Fader, S. Bellman, and G. L. Lohse. On the depth and dynamics of online search behavior. *Management Science*, 50(3):299–308, 2004.

[15] R. Jones and D. Fain. Query word deletion prediction. In *Proc. of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 435–436, 2003.

[16] R. Jones and K. L. Klinkner. Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In *Proc. of the 17th ACM Conference on Information and Knowledge Management*, pages 699–708, 2008.

[17] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proc. of the 15th International Conference on World Wide Web*, pages 387–396, 2006.

[18] T. Lau and E. Horvitz. Patterns of search: Analyzing and modeling web query refinement. In *Proc. of the 7th International Conference on User Modeling*, pages 119–128, 1999.

[19] A. L. Montgomery and C. Faloutsos. Identifying web browsing trends and patterns. *IEEE Computer*, 34(7):94–95, 2001.

[20] H. C. Ozmutlu. Automatic new topic identification using multiple linear regression. *Information Processing and Management*, 42(4):934–950, 2006.

[21] H. C. Ozmutlu and F. Cavdur. Application of automatic topic identification to Excite web search engine data logs. *Information Processing and Management*, 41(5):1243–1262, 2005.

[22] Y.-H. Park and P. S. Fader. Modeling browsing behavior at multiple websites. *Marketing Science*, 23(3):280–303, 2004.

[23] F. Radlinski and T. Joachims. Query chains: Learning to rank from implicit feedback. In *Proc. of the 11th International Conference on Knowledge Discovery and Data Mining*, pages 239–248, 2005.

[24] A. Spink, M. Park, B. J. Jansen, and J. Pedersen. Multitasking during web search sessions. *Information Processing and Management*, 42(1):264–275, 2006.

Kosmix: Exploring the Deep Web using Taxonomies and Categorization

Anand Rajaraman
Kosmix Corporation, Mountain View, CA, USA
anand@kosmix.com

Abstract

We introduce topic exploration, a new approach to information discovery on the web that differs significantly from conventional web search. We then explain why the Deep Web, an inhospitable region for web crawlers, is emerging as a significant information resource. Finally, we describe the anatomy of Kosmix, the first general-purpose topic exploration engine to harness the Deep Web. The Kosmix approach to the Deep Web leverages a huge taxonomy of millions of topics and their relationships, and differs significantly from that adopted by web search engines such as Google.

1 Introduction

Web search engines, such as those developed by Google, Yahoo, and Microsoft, excel at finding the needle in a haystack: a single fact, a single definitive web page, or the answer to a specific question. Often, however, the user's objective is not to find a needle in a haystack, but to learn about, explore, or understand a broad topic. For example:

- A person diagnosed with diabetes wants to learn all about this disease. The objective is not just to read the conventional medical wisdom, which is a commodity available at hundreds of websites, but also to learn about the latest medical advances and alternative therapies, evaluate the relative efficacy of different treatment options, and connect with fellow-sufferers at patient support groups.
- A reporter researching a story on Hillary Clinton needs access to her biography, images, videos, news, opinions, voting record as a lawmaker, statements of financial assets, cartoons and other political satire.
- A traveler planning a trip to San Francisco needs to learn about attractions, hotels, restaurants, nightlife, suggested itineraries, what to pack and wear, and local events.

These are just three of numerous use cases where the goal is to explore a topic. Topic exploration today is a laborious and time-consuming task, usually involving several searches on conventional web search engines. The problem in many cases is knowing exactly what to search for; in the diabetes example above, if the diabetes

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

sufferer knows there are patient-support groups, or that there might be alternative therapies they might be interested in, it's not hard to find them through conventional search engines. The bigger issue is that most people exploring a topic don't know what to look for.

Kosmix tackles this problem by creating a *topic page* for any topic. The goal of a topic page is to provide a 360-degree view of a topic. In each of the examples above, the Kosmix topic page includes all the information listed. In addition, the topic page also provides a list of related topics, conveniently grouped together, that suggest further areas to explore. The topic page uses a two-dimensional layout that is more reminiscent of a newspaper or a magazine than a search results page. This is not a coincidence. Magazines and newspapers, like topic pages, have been designed for browsing as opposed to search; the goal is to facilitate serendipitous information discovery.

In order to validate the hypothesis that topic pages are valuable for certain use cases, the Kosmix team in 2007 launched RightHealth, a topic exploration engine for health information. In April 2009, the site drew over 7 million unique visitors and served over 20 million topic pages, making it the number 2 health information website in the US according to Hitwise, an independent measurement agency. Having validated the usefulness of topic pages, the Kosmix team in December 2008 launched kosmix.com, extending the RightHealth model to topic pages across all categories. By April, just 4 months after launch, traffic to the new site has grown to over 3 million unique monthly visitors.

A second distinction between Kosmix and search engines lies in the source of the information on the Kosmix topic page. Search engines construct their search results pages from their indexes, which in turn are built using web crawlers. A Kosmix topic page consists of a collection of modules. Each module is constructed using the result of an API call to a web service, done at the time of page construction. This allows Kosmix to tap into the Deep Web, the portion of the web not accessible to crawlers: social networking sites, media-sharing sites (photos, videos, documents), library catalogs, airline reservation systems, phone books, scientific databases and all kinds of information that lie concealed from view behind web forms. Some estimates have pegged the size of the Deep Web at upto 500 times larger than the Surface Web [13].

Although there are differences in the ranking algorithms used by different search engines, the fundamental architecture of web search engines is well-understood [2]. Of late, there has also been some interest in adding Deep Web data to search engines using a enhanced web crawler [8]. A few domain-specific vertical search engines, such as Mobissimo and Kayak, have adopted a federated search approach to the Deep Web, accessing deep web sources at query-time and constructing results pages based on their responses. This paper describes the anatomy of the Kosmix explore engine, which to our knowledge is the first general-purpose, domain-independent service that uses a hybrid of the two approaches to harness the Deep Web.

The rest of this paper is organized as follows. Section 2 introduces the Deep Web and explains some of the industry trends that make the Deep Web more relevant by the day. Section 3 describes the two approaches to harnessing the power of the Deep Web and evaluates their advantages and disadvantages. Section 4 describes the anatomy of the Kosmix explore engine. Section 5 describes related work. Section 6 concludes with some thoughts on the evolution of the Deep Web.

2 The Deep Web

While the Deep Web has existed for almost as long as the Surface Web, a confluence of industry trends has made it more relevant today than at any time in the past.

User-generated content and social media. The Web 2.0 revolution has seen an explosion of sites dedicated to user-generated content (UGC). Examples include Wikipedia, YouTube, Flickr, and specialized sites in many subject areas such as TripAdvisor and Yelp. Such sites have removed friction from the content-creation process and lead to a sharp increase both in the number of contributors and in content of various media types. While search engines have strived to keep up with the deluge, the amount of information available on such services is growing faster than search engine index sizes. Social media sites such as Facebook, MySpace, and Twitter take this trend even further. The information on such sites is subject to various access controls. For example, users may permit only their “friends” to view certain information. In such cases, web crawlers may not have access to the information, since they don’t work on behalf of a real user.

Real-Time. One of the weaknesses of the conventional web search architecture is the time lag introduced by the crawl and index process. While search engines have reduced this latency considerably over the past years, it is still a problem for information of a time-sensitive nature. Examples include ticketing and reservation systems of all kinds, auctions and shopping sites with limited product availability, and financial information related to the stock market.

One example that has captured the popular imagination of late is Twitter. When breaking-news events happen, Twitter users are often the first to post news and images online, often within seconds. Instances of this kind include the earthquake in China (May 2008) and the landing of US Airways 1549 on the Hudson River (February 2009). In such cases, there is a window of time where Twitter Search produces superior results over any conventional web search engine.

Specialized search engines. Search ranking algorithms such as PageRank evolved at a time when the web consisted primarily of hyperlinked HTML documents. However, a large fraction of useful content available today does not fit the old model and requires different ranking methodologies. For example, media-sharing sites such as YouTube and Flickr have access to information such as the reputation of the person who uploaded the content, number of views, and ratings. Another trend is the emergence of specialized search engines for specific tasks, such as Mobissimo and Kayak for travel; SimplyHired for job listings; Shopping.com and TheFind.com for products; and so on. As the web evolves from static documents to dynamic content repositories, it would seem that the most natural approach is to let such sites develop their own site-specific search engines, and then federate the results of these engines, or of domain-specific aggregators.

Information presentation. The results presentation model of web search engines has not evolved significantly in over 10 years. In the meantime, specialized services (either site-specific search engines, or category-specific aggregators) have developed innovative information presentation models for specific use cases. For example, Zillow for home prices; TheFind for products; Facebook for user profiles; and so on. Once again, a federated approach makes a lot of sense. Yahoo’s SearchMonkey effort is a first step by search engines to address this issue. It allows content owners to submit a catalog of *rich snippets* for a set of queries. The limitation of this aproach is that the set of queries needs to be specified in advance, together with a static snippet for each query.

Availability of APIs. A growing trend is the evolution of websites into web services. Web services provide access to the contents and capabilities of the site via APIs. A handful of standards have evolved for such APIs, including REST and JSON. Crucially, we have reached a tipping point where the quantity of useful information available through such APIs is sufficient to build useful services.

Business model issues. The dominance of search engines as gateways to the web tends to commoditize web content and intermediate between content creators and consumers. Many content creators are therefore wary of allowing search engines access to their entire data set. For example, Twitter does not provide search engines except Twitter Search access to its API. Facebook allows only limited access to search engines. Several newspapers have digitized archives that date back several decades, but do not make these archives available to search engine crawlers. Record labels do not make copyrighted music available for crawl. Such information can in many cases be accessed only through an API.

Infinite Wine in a Finite Bottle. Content created by algorithms is an interesting new trend.The beginnings of this trend can bee seen in the Wolfram Alpha, which can answer questions that are mathematical computations over data. Since the set of mathematical computations is infinite, trying to represent the Alpha as a finite set of web pages to fit in an index is a futile exercise. An API is the only sensible way to access such sites.

3 Approaches to Deep Web Search and Exploration

There are two fundamentally different approaches to incorporating the deep web into search or topic exploration engines.

- **Deep Web Crawl.** Crawl as much of the deep web as possible and incorporate it into a conventional search engine index.
- **Federated Search.** Use APIs to access deep web sources at query-time and construct results pages based on their responses.

Wright [13], using a fishing analogy, calls these approaches *trawling* and *angling* respectively, while Madhavan et al. [8] calls them *surfacing* and *virtual integration.* These approaches are also analogous to the warehousing and mediation approaches in data integration.

3.1 Deep Web Crawling

The crawl-based approach has the advantage that it fits well with the conventional web search model. The additional documents can be added to the search index and ranked using the existing ranking algorithm.The deep web crawl approach has been employed very effectively at Google, especially for tail queries [8].

The disadvantages of the deep web crawl approach relate to many of the issues described in Section 2. In particular, the deep crawl approach shares many of the shortcomings of the crawl-index-search architecture with respect to indexing social media sites, real-time data, specialized search algorithms and presentations for different data types, business model issues, and the problem of services that can handle infinite query sets.

3.2 Federated Search

The dynamic query approach overcomes many of the limitations of the crawl-index-search approach outlined in Section 2. However, the approach comes with its own set of challenges.

Integrating APIs. Each web service comes with its own API, which uses different parameters from every other API. The results are also formatted differently. A piece of custom code, conventionally called a wrapper, is required to connect to each web service, limiting the scalability of the approach.

Source Selection. Given a topic and thousands of potential information sources, it is not practical to query every information source for each query. For example, it is not meaningful to send the query "diabetes" to TripAdvisor.com, a travel resource. If we indiscriminately query a source for topics irrelevant to it, we run into two issues. The first is the potential to clutter the results page with irrelevant information. The second is overloading web service providers, potentially bringing down their systems or getting them to reject future queries from the explore engine. Unlike in the case of an index-based search engine, source selection needs to be done without having access the content of the source.

Query Transformation. Kosmix users enter free form text queries, while the underlying information sources often support a richer query model. Once we deem an information source as potentially useful for a query, the next task is to rewrite the query in the manner most suited for that source. It should be noted that this problem is very different from that of rewriting structured queries, which has been explored in detail in the context of information integration [11, 7].

Results Layout. As we discussed in Section 2, many data sources present results using innovative methods tailored to the kind of data, and we would like to preserve this richness rather than degenerate to the lowest common denominator. In addition, since results are inherently of different types, it is unnatural to force a linear ranking across them.

Performance. Unlike a web search engine, a federated search engine needs to make external API calls in order to construct its pages. Therefore, we should expect this approach to be inherently slower than web search.

Long response times can, however, have the effect of turning off users. Therefore, it is a huge implementation challenge to keep response times within acceptable limits.

4 The Kosmix Explore Engine

In this section we describe the overall architecture and various design decisions that have gone into the Kosmix Explore Engine. We describe in some detail the Kosmix approach to data source access, source selection, and results layout. Our approach to query rewriting and caching (to tackle the performance challenge described in the previous section) will be described in a companion paper.

4.1 A Hybrid Approach to Data Source Access

In the previous section, we listed some of the pros and cons of the crawl-based and federated search models. In practice, Kosmix uses a hybrid approach that combines features of the crawl and the federated search approaches. Some of the data is indexed locally, while API calls are made in other cases.

One of the modules that is surfaced on every topic page is a web search results module, with results from a leading web search engine (Google). In effect, users get the best of both worlds: the comprehensiveness of conventional web search, with all the other advantages of the federated search model. Since the web search module ensures comprehensiveness, we have chosen to focus only on Deep Web data sources that cannot be handled satisfactorily by the surfacing approach described in [8], which is already incorporated into the web search module. Such data sources fall into a few categories:

- Data sources that cannot be adequately crawled using surfacing techniques, either because of their size (e.g., Flickr), business model reasons (e.g., newspaper archives, music), access permissions (e.g., Twitter, Facebook), or because they are computation engines (e.g., Wolfram Alpha).
- Data sources that use specialized search algorithms (e.g., YouTube, Flickr, Truveo).
- Data sources that present results in unique ways (e.g., TripAdvisor, Yelp).
- Data sources where the data varies rapidly with time (e.g., Twitter, airline ticketing).

Data sources that meet any one of the above criteria are candidates for the federated API approach. At Kosmix we have identified several thousand such services, and have integrated thousands of them into the explore engine. We have developed a tool, called Modulator, that supports the common types of web APIs (e.g., REST, JSON) and makes it a relatively quick task to add a new service. It should be noted that adding a data source to Kosmix is a much simpler task than writing a wrapper for a data integration system, because the explore engine does not need to interpret the results from an API call, which is usually the time-consuming part of creating wrappers.

There is also a class of data sources that are not, strictly speaking, Deep Web sources at all, but have interesting structure that makes it unsatisfactory to view them as just collections of web pages. Examples of such sources include Wikipedia, Chrome (data about various car makes and models), A.D.A.M. (doctor-reviewed summaries of many diseases and conditions), and IMDB (the Internet Movie Database). We have chosen to locally index such databases using a structured data index. Once indexed in this fashion, these structured indexes can be treated as identical to external APIs, only with much higher performance.

4.2 The Taxonomy

The cornerstone of the Kosmix explore engine is its taxonomy and categorization technology. The Kosmix taxonomy consists of millions of topics organized hierarchically, reflecting is-a relationships. For example, San

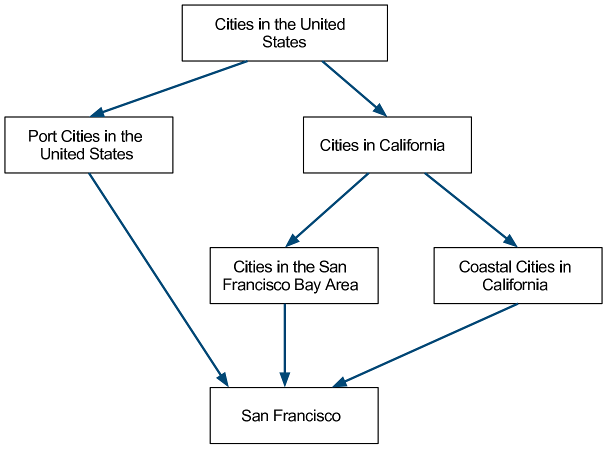
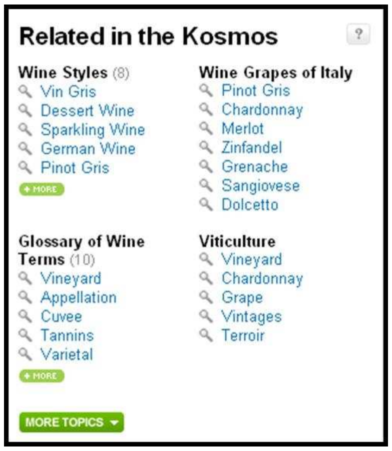

Cities in the United States
Port Cities in the United States
Cities in California
Cities in the San Francisco Bay Area
Coastal Cities in California
San Francisco

Figure 1: A small fragment of the Kosmix taxonomy


Related in the Kosmos

Wine Styles (8)
- Vin Gris
- Dessert Wine
- Sparkling Wine
- German Wine
- Pinot Gris

+ MORE

Wine Grapes of Italy
- Pinot Gris
- Chardonnay
- Merlot
- Zinfandel
- Grenache
- Sangiovese
- Dolcetto

Glossary of Wine Terms (10)
- Vineyard
- Appellation
- Cuvee
- Tannins
- Varietal

+ MORE

Viticulture
- Vineyard
- Chardonnay
- Grape
- Vintages
- Terroir

MORE TOPICS

Figure 2: Topics related to Pinot Noir

Francisco is-a city. The resulting hierarchical structure is a directed acyclic graph (DAG). Figure 1 shows a small piece of the Kosmix taxonomy.

The taxonomy also encodes many relationships beyond is-a. For example, there is a member-of relationship connecting a music group with its members, and a capital-of relationship connecting a country with its capital city. There are many thousands of such relationship types captured in the taxonomy.

The taxonomy has been built over three years using a combination of human curation and algorithmic methods. The raw materials include several publicly available taxonomies, such as DMOZ and Wikipedia, as well as hundreds of special purpose taxonomies in specific fields, such as health, automobiles, and music. The details of how the taxonomy is created and maintained need not concern us here, but the technical challenges we had to surmount include:

- Merging overlapping taxonomies, taking into account that the same concept might be named differently in the two taxonomies.
- Keeping the taxonomy up to date by identifying new topics on an ongoing basis. At Kosmix we gather and analyze millions of RSS feeds every day to identify new topics, such as people, music groups, and so on.
- Adding new relationships between existing topics in the taxonomy.

4.3 Categorization

The second key to the Kosmix explore engine is the Kosmix Categorization Service (KCS). Given a user query, KCS determines the nodes in the taxonomy that are most closely connected with the query. The details of the algorithms involved are proprietary to Kosmix and are not relevant to the discussion here. We will content ourselves with an example illustrating the functionality provided by KCS.

Let us say the query is “Pinot Noir.” KCS determines that Pinot Noir is a kind of wine, which is a related to foods and beverages. It also determines that Pinot Noir is a kind of wine grape, and is related to viticulture and vineyards. Figure 2 shows a small selection of the full list of topics KCS determines are related to Pinot Noir. These topics are displayed on the topic page in the section titled *Related in the Kosmos.*

4.4 Source Selection

When a data source is added to the explore engine, it is also associated with nodes in the taxonomy. For example, the TripAdvisor data source is associated with the Hotels node, while Yelp is associated with Restaurants, and WebMD with Health. The Modulator tool mentioned earlier has algorithms that help a human make these associations.

At query time, the explore engine first invokes KCS to determine taxonomy nodes that are related to the query. It then ranks data sources based on how closely they are associated with the query in the topic-space. As one example of a ranking function, we can imagine both the query and data sources as vectors in the topic space, and use a cosine distance measure. In practice, the ranking function used by the explore engine is much more sophisticated, taking into account is-a and other relationships in the taxonomy.

Continuing with the Pinot Noir example, sites deemed relevant to this topic include Epicurious and Food Network (food and recipe-related websites), DailyPlate and FatSecret (both databases listing nutritional value of food), and Amazon.com (for wine shopping). In addition, there are also many general-purpose services that have content on a wide variety of topics, including Wikipedia, Google Image Search, and YouTube.

Once the explore engine identifies a number of candidate data sources (say 15-20) relevant to the topic, it sends the user query to each source. An important caveat is that while the data source may be identified as a candidate for a query, it is not guaranteed to have good results for that query. Therefore, the results from each candidate source are post-processed and then a second ranking function based on the actual returned results is used to determine the final source ranking.

4.5 Laying out the Topic Page

In Section 4.4 we described how the explore engine computes a linear ranking of data sources for a query. The results, however, are not laid out in a linear fashion, but grouped together by information type into a 2-dimensional layout: text, audio, video, user profiles, shopping, conversations and so on. The page real estate allotted to a group varies based on various factors, such as the relevance scores of the data sources in the group. Within a group, each data source gets a variable amount of real estate depending upon its relevance and other factors.

The *Related in the Kosmos* module enables exploration by surfacing topics in the taxonomy that are related to the query, grouped in a fashion that makes it easy to to scan. Figure 2 shows some of the related topics for the query "Pinot Noir." Clicking on one of these links takes the user to the page for that topic.

5 Related Work

Broder [3] and several other works have classified search queries into three categories: *navigational*, *informational*, and *transactional*. While the exact proportions have varied, there is agreement that a large fraction of queries are informational. Topic exploration is a good metaphor for informational queries.

There is a significant body of work on creating vertical search engines for specific domains by constructing semantic mappings from a mediated schema (or form) to collections of forms within a domain [4, 7, 12, 15]. Most of this work assumes that the mediated schema can be created by hand. Our work, on the other hand, focuses on a general-purpose, domain-independent explore engine. The sheer breadth of queries we need to handle makes it impossible to manually create a mediated schema, leading us to the automated creation and maintenance of a taxonomy.

Google's approach to crawling the Deep Web is described in [8]. There has been prior work around acquiring documents from databases with restricted query interfaces, as well as computing keyword distributions that summarize database contents to facilitate source selection [1, 5, 6, 9, 10, 14]. There has been significant research on query transformation and rewriting in the context of data integration [7, 11].

6 Conclusion

We described the anatomy of Kosmix, the first general purpose Deep Web Explore Engine. Kosmix enables a new approach to information finding, called topic exploration. We illustrated several use cases satisfied by this model that are not served satisfactorily by today's web search engines. Kosmix uses a hybrid approach to the Deep Web that combines elements of the crawl and federated search approaches. Traffic to Kosmix and its specialized health property RightHealth continue to increase at a rapid clip, demonstrating the value of the Kosmix approach to deep web exploration.

The architecture of web search engines today reflects the historical dominance of the Surface Web. The increasing importance of the Deep Web is forcing a rethink of this architecture. The hybrid approach to the Deep Web has several advantages over both the crawl-only and pure federated approaches. Some standardization around web service APIs could dramatically increase the scalability of this approach. For example, there could be some basic standards around names of input fields such as location-related inputs, text search inputs, category inputs and so on. Even this small amount of standardization can take Deep Web exploration to the next level.

Acknowledgements

The Kosmix explore engine described in this paper is the collective effort of the Kosmix team.

References

[1] L. Barbosa and J. Freire. Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, 2004.

[2] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *WWW7*, 1998.

[3] A. Broder. A taxonomy of web search. In *SIGIR Forum*, 36(2):3-10, 2002.

[4] A. Doan, P. Domingos, A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD*, 2001.

[5] L. Gravano, P. G. Iperiotis, M. Sahami. QProber: A system for automatic classification of deep-web databases. *ACM Transactions on Information Systems*, 21(1):1-41, 2003.

[6] P. G. Iperiotis and L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In *VLDB*, 2002.

[7] A.Y. Levy, A. Rajaraman, J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, 1996.

[8] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, A. Y. Halevy. Google's Deep Web crawl. *PVLDB* 1(2): 1241-1252, 2008.

[9] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden-web content through keyword queries. In *JCDL*, 2005.

[10] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *VLDB*, 2001.

[11] A. Rajaraman, Y. Sagiv, and J.D. Ullman. Answering Queries using Templates with Binding Patterns. In *PODS*, 1995.

[12] J. Wang, J.-R. Wen, F. Lochovsky, and W.-Y. Ma. Instance-based schema matching for web databases by domain-specific query probing. In *VLDB*, 2004.

[13] A. Wright. Searching the Deep Web. In *CACM*, 51(10):14-15, October 2008.

[14] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma. Query selection techniques for efficient crawling of structured web sources. In *ICDE*, 2006.

[15] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *SIGMOD*, 2004.

Flexible Querying of Personal Information

Amélie Marian, Wei Wang

Department of Computer Science, Rutgers University, New Brunswick, NJ, USA

{amelie,ww}@cs.rutgers.edu

Abstract

The amount of personal data that users store and access in personal information systems has grown massively. Simple flexible search tools are a necessity for users to be able to access the data they need. In addition, the file organization that is typical in file systems is not a practical logical view anymore as applications may store information across files, or combine different objects into a single file. Users cannot be expected to know or remember the exact location and specific details about the data they are looking for; search techniques should allow for some approximation during query processing in order to return useful results while accounting for possible errors in the query. We present a scoring framework that takes into account approximation in both the content and structural dimensions of the data. Our query model extends the search to consider information across file boundaries. We implemented a prototype of our search techniques over a real data set and report on our experimental results.

1 Introduction

The amount of data stored in personal information management systems is rapidly increasing, following the relentless growth in capacity and the dropping price per byte of storage in personal computing systems. This explosion of information is driving a critical need for complex search tools to access often very heterogeneous data in a simple and efficient manner. Such tools should provide both *high-quality* scoring mechanisms and *efficient* query processing capabilities.

Numerous search tools have been developed to perform keyword searches and locate personal information stored in file systems, such as the commercial tools Google Desktop [10] and Spotlight [14]. However, these tools usually index text content, allowing for some *ranking* on the textual part of the query—similar to what has been done in the Information Retrieval (IR) community—but only consider structure (e.g., file directory) and metadata (e.g., date, file type) as *filtering* conditions. Recently, the research community has turned its focus on search over to Personal Information and Dataspaces [4, 6, 8], which consist of heterogeneous data collections. However, as is the case with commercial file system search tools, these works focus on IR-style keyword queries and use other system information only to guide the keyword-based search.

Keyword-only searches, while reasonably efficient in practice, consider files as bags of words and do not exploit the rich structural information that is typically available in personal information systems. Unlike searches over digital libraries and the Web, users searching their personal files possibly have in-depth knowledge of where they expect the file to be located (directory structure), of details such as file type and creation date on the file

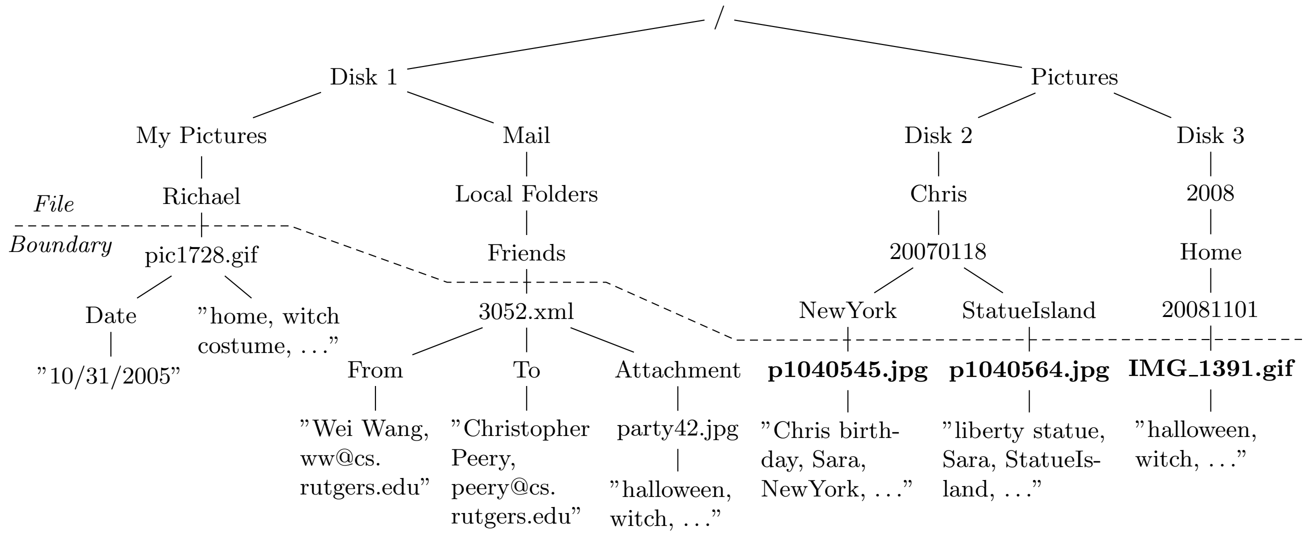


Figure 1: A subset of a user personal information file structure.

itself (meta-data structure), and of complex structural information contained within the file (internal structure). Internal structure can be derived from the file format, e.g., email files `<from>` and `<to>` fields, or could consist of annotation data associated with the files, e.g., tags given to photo files. Using this structural information only as filtering conditions during query processing is too rigid because any mistake in the query will lead to relevant files being missed; a flexible approach that allows for some error in the structural conditions is desired, as illustrated by the following example:

Example 1: Consider Chris, a user saving personal information in the file system of a personal computing device. Chris wants to retrieve photos of a Halloween party that was held at his home where someone was wearing a witch costume. Unfortunately, Chris does not remember in which year that party tool place.

Ideally, the file directory structure would have been created and maintained consistently and all photos properly tagged. In real-life scenarios, this is rarely the case: users change their file organizations over time, inconsistently annotate their data and may gather information from multiple places. In our example, Chris has changed the way he organizes his photos over time when he switched to a new computer and decided to use a new photo organization software, and his pictures are not consistently tagged. As a result, pictures from Halloween parties held in different years match very different directory structures and do not necessarily have matching tags, as illustrated in Figure 1. In addition, some relevant pictures are not in Chris's main photo hierarchy, but in his email folder as they were sent to him by friends who attended the party.

This structural heterogeneity complicates the search for specific pictures. A content-only search for "Halloween, home, witch," even considering only pictures, is likely to result in many matches, of various relevance. None of the pictures contained in the example data set of Figure 1, contain all three keywords. Three contain two of the keywords; their relative rankings would depend on the underlying content scoring function. One of the three picture, IMG_1391.gif is however arguably the best match as its directory structure contains the third missing keyword.

Another possibility is to write a more refined query that takes into account the directory structure as well as the structural metadata information:

```
[filetype=⋆.gif AND
content="witch" AND
structure=/home/Halloween]
```

Current tools would answer this query by returning all files of type **.gif* under the directory */home/Halloween* (filtering conditions) that have content similar to "witch" (ranking expression). Because the directory structure, and file type are used only as filtering conditions, only the exact match IMG_1391.gif would be returned as a result to this query, files that are very relevant to the content search part of the query, but which do not satisfy these exact conditions would not be considered as valid answers. For example, the file pic1728.gif, which contains the keywords "witch" and "home" but is not in the query target directory would not be returned although it may be a suitable approximate match to the query.

Because of the nature of personal information systems, we believe it is critical to support approximate matches on both the content and structural components of the query. In previous work [11, 12], we presented a scoring framework that considers relaxed query conditions on several query dimensions. Our approach individually scores the content, metadata, and structure dimensions and combines the resulting scores in a unified scoring framework using an *IDF*-based interpretation of scores for each dimension. We developed efficient and dynamic index structures to support our scoring techniques. In this paper, we extend our query model to consider structure within the file and relax structural conditions across the file boundaries. We describe our data and query model in Section 2 and present preliminary results in Section 3. We discuss open research issues and future work directions in Section 4.

2 Data and Query Model

As shown in Figure 1, our personal information data model considers structure both outside and within the file in a unified manner. The whole file system can be seen as an XML document, with the content of the file stored in leaf nodes.

Our system currently supports relaxed query conditions on three dimensions: *content* for conditions on the text content of files, *metadata* for conditions on the system information about files, and *structure* for conditions on both the directory paths to access files and the internal structure of the file. In this paper, we focus our discussion on the structure dimension. Our relaxed queries can be viewed as simple XQuery [15] expressions. As a first step, we only consider physical files as possible query results. We plan to relax this constraint and allow for various levels of granularity of query results.

Any file that is relevant to one or more of the query conditions is then a potential answer for the query; it gets assigned a score for each dimension based on how close it matches the corresponding query condition. Scores across multiple dimensions are unified into a single overall score for ranking of answers.

Our scoring strategy was presented in [12] and is based on an *IDF*-based interpretation of scores, as introduced in [1] and described below. For each query condition, we score files based on the *least relaxed* form of the query that each file matches. Scoring along all dimensions is uniformly *IDF*-based which permits us to meaningfully aggregate multiple single-dimensional scores into a unified multi-dimensional score. Our structure scoring strategy extends prior work on XML structural query relaxations [2]. In particular, we use several types of structural relaxations, some of which were not considered in [2], to handle the specific needs of user searches in a file system. Assuming that structure query conditions are given as pathnames, these relaxations are:

- **Edge Generalization** is used to relax a parent-child relationship to an ancestor-descendant relationship. For example, applying edge generalization to $/home/Halloween$ would result in $/home//Halloween$.
- **Path Extension** is used to extend a path P such that all files within the directory subtree rooted at P can be considered as answers. For example, applying path extension to $/home/Halloween$ would result in $/home/Halloween//*$.
- **Node Deletion** is used to drop a node from a path. For example, applying node deletion on $Halloween$ from $/home/Halloween/witch$ would result in $/home//witch$.

- **Node Inversion** is used to permute nodes within a path. For example, applying node inversion on $Halloween$ and $witch$ from $/home/Halloween/witch$ would result in $/home/(Halloween/witch)$, allowing for both the original query condition as well as $/home/witch/Halloween$.

- **Node Extension** is used to allow for structural conditions to be matched by the content information contained within the files. For example, applying node extension on $/home/Halloween$ would result in the query $/home/\{Halloween\}$, whose meaning is that the term “Halloween” could be part of the external directory structure, internal file structure, or file content (represented as leaf nodes in our unified structure)

We then say that a file *matches* a (possibly relaxed) query condition if all structural relationships between the condition’s components are preserved in the file’s unified external and internal structure.

Finally, given a directory structure D and structure query Q, the structure score of a file f with respect to Q is computed as:

$$score_{Structure}(Q, f) = \max_{P \in R(Q)} \{score_{idf}(P) | f \in F(D, P)\} \tag{1}$$

$$score_{idf}(P) = \frac{log(\frac{N}{N_P})}{log(N)}, \quad N_P = |F(D, P)| \tag{2}$$

where $R(Q)$ is the set of all possible relaxations of Q, $F(D, P)$ is the set of all files that match a relaxation P of Q or P’s extension in D, and N is the total number of files in D.

We represent all possible relaxations of a query condition, along with the corresponding *IDF* scores for (files that match) each relaxation, using a DAG structure. The DAG is created by incrementally applying query relaxations to the original query condition, the top level (root) of the DAG is a single node that represents the original exact query. Children nodes of a DAG node are more relaxed versions of the query condition and therefore match at least as many answers as their parents (containment property). The *IDF* score associated with a DAG node can be no greater than the score associated with its parents [1, 12]. As we expand the DAG and traverse it further away from the root, the increasingly relaxed versions of the query conditions match more and more files resulting in lower *IDF* scores. The most relaxed version of the query condition: *//** matches all files and it has a score of 0.

Efficiently processing relaxed queries across file boundaries raises several challenges:

- The set of all possible relaxations is query-dependent and the size of the DAG grows exponentially with the query size, i.e., the number of path nodes in the query. As such it must be dynamically and lazily built at query time, and efficient index building and traversal techniques are critical issues.

- Inverted indexes for fast access to the structure terms should allow to seamlessly integrated all types of structural data (external, internal, metadata), which are stored separately by current file systems implementation.

- Query processing algorithms should be adapted to handle the proposed relaxations in an efficient manner.

We have adapted the Threshold Algorithm (TA) [7] to our scenario. *TA* uses a threshold condition to avoid evaluating all possible matches to a query, focusing instead on identifying the k best answers. To efficiently evaluate potential matches to flexible structure queries, we have extended the PathStack algorithm [3] to handle our proposed relaxations. In particular, our implementation allows for *node inversion*. We report on our evaluation results in the next section.

3 Experiments

We now present our preliminary experimental results for our relaxed structure scoring techniques that span across the file structure boundaries.

3.1 Experimental Settings

Platform. We evaluate the performance of our search approach on a prototype implemented in Java. We use the Berkeley DB [13] to persistently store all indexes. Experiments were run on a PC with a 64-bit hyper-threaded 2.8 MHz Intel Xeon processor, 2 GB of memory, and a 10K RPM 70 GB SCSI disk, running the Linux 2.6.16 kernel and Sun's Java 1.5.0 JVM. Reported query processing times are averages of 40 runs, after 40 warm-up runs to avoid measurement JIT effects. All caches (except for any Berkeley DB internal caches) are flushed at the beginning of each run.

Data set. As noted in [6], there is a lack of synthetic data sets and benchmarks to evaluate search over personal information management systems. Thus, we use a data set that contains files and directories from the working environment of one of the graduate student involved in the project. We focus our preliminary experiments on searches for photo files. The data set contains 2.8 GB of photo data from 3,437 picture files organized in 216 directories. The average directory depth was 4 with the longest being 6. On average, each directory contains 16.9 sub-directories and files, with the largest containing 91. The prototype extracted 27,319 unique stemmed content and directory terms.

Techniques. We compare the performance of our flexible structural search approach (*Structure*) with that of the standard Information Retrieval content-only approach (*Terms*). We use the popular Lucene package for the content indexing. Finally, we also report on two extensions of the content-only *Terms* technique that search the content terms of the file as well as the directory structure terms, viewed as a "bag of words," either separately or combined.

3.2 Experimental Results

Our study focuses on the performance of the techniques when searching for specific target photo files. The three target files, along with their corresponding directory structures and content tags, are highlighted in Figure 1.

For each target file, we construct various structural and content queries aimed at retrieving the file. The results are shown in Table 1. Queries 1 to 12 target p1040564.jpg (Figure 1), a picture taken on Chris's birthday with Sara in New York, queries 13 to 19 target IMG_1391.gif, a picture of a witch costume taken at home on Halloween, and queries 20 to 26 target p1040545.jpg, a picture of Sara taken at Liberty Statue island. We constructed queries that correspond to realistic user searches for these target photos.

In particular, we constructed (a) several structural queries (Q1-5, Q13-15, Q20-21) that use the *Structure* technique described in Section 2 to search the unified structure considering both information in the directory structure and within the file; (b) content queries that only searches the file content (Q6, Q16, Q23), (c) queries that consider the file content and directory terms separately (Q7-11, Q17-18, Q24-25), and (d) queries that consider the file content and directory terms together, i.e., that augment the content index of each file with the terms contained in its directory path, (Q12, Q19, Q26).

As shown in Table 1, queries in category (a) using our flexible structural search return the target file as the top answer for all three cases. (A range in the Rank column indicates the presence of ties.) This is true regardless of the error in the original query structure; our relaxation approach allows to capture the correct target even if the user has entered incorrect information, as long as the query terms appear either in the file content or in its directory path. Content-only queries from category (b) failed to rank the target files at the top as they will not take into account a query term that appears in the directory path. In effect, this leads to many irrelevant file that contain some, but not all, query terms to be pushed to the top of the query result. Queries from category (c)

Query Evaluation Results						
		Query Conditions				
		FlexStructure	Terms			
Target	Query		File	Dir	File+Dir	(Rank)
p1040564.jpg	Q1	/c[./n and ./s and ./b]	-	-	-	1-2
	Q2	/b[./c and ./n and ./s]	-	-	-	1-2
	Q3	/n[./c and ./s and ./b]	-	-	-	1-4
	Q4	/s/b[./c and ./n]	-	-	-	1-2
	Q5	/c/n/s/b	-	-	-	1-2
	Q6	-	c,n,s,b	-	-	14
	Q7	-	n,s,b	c	-	1-2
	Q8	-	c,n,b	s	-	133-137
	Q9	-	c,s,b	n	-	3-4
	Q10	-	c,n,s	b	-	293-294
	Q11	-	c,n	s,b	-	35-39
	Q12	-	-	-	c,n,s,b	10-11
IMG_1391.gif	Q13	/h[./w and ./a]	-	-	-	1
	Q14	/a[./h and ./w]	-	-	-	1
	Q15	/a/h/w	-	-	-	1
	Q16	-	a,h,w	-	-	19
	Q17	-	w,a	h	-	1
	Q18	-	h,w	a	-	166
	Q19	-	-	-	a,h,w	19
p1040545.jpg	Q20	/i[./"s" and ./"l"]	-	-	-	1
	Q21	/s[./"i" and ./"l"]	-	-	-	1
	Q22	/s/i/l	-	-	-	1
	Q23	-	i,s,l	-	-	33
	Q24	-	s,l	i	-	1
	Q25	-	i,l	s	-	151-159
	Q26	-	-	-	i,s,l	19

Table 1: The rank of target files returned by a set of structural and content queries. Content and structural values are abbreviated. The complete terms are Chris (c), birthday (b), Sara (s), NewYork (n), Witch (w), home (h), Halloween (a), Liberty (l), and StatueIsland (i). A range of values for Rank means there are ties in scores.

consider the content and directory term separately and only perform well if the user did not make any errors in identifying the location of the term, i.e., if the query is close to an exact query. An error in this setting is particularly costly and likely results in the target file not being found. Considering the content and directory terms together, as is done by queries in category (d), leads to more consistent results, but the target file does not appear in the top-10 answers.

These results demonstrate that our flexible structural search approach has the potential to significantly improve search accuracy over existing approaches based on "bag of words" approaches. Specifically, it captures the structural relationships given in the query to improve search results, yet is very robust against potential query errors.

Figure 2 reports on the query performance of the queries of Table 1. As shown in the figure, our flexible structure scoring strategy has an overhead over the term-based strategies. This overhead is due to the construction of the structure DAG at query evaluation time. We have implemented a lazy DAG traversal strategy as well as several query processing optimizations on the DAG structure, the time cost reported in Figure 2 include these optimizations. Interestingly, queries from category (c), which have the potential to have high-quality answers, exhibit evaluation times comparable to our flexible structure scoring as there is an overhead incurred when combining the separate content and directory terms scores.

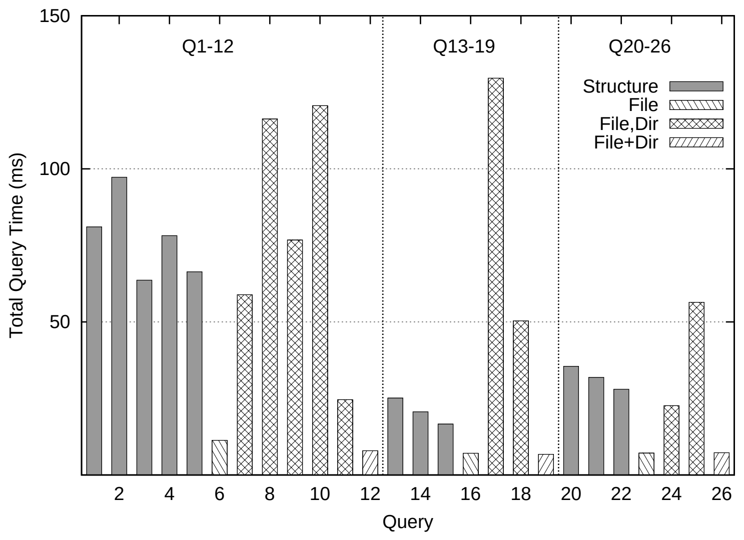


Figure 2: Query Processing time.

This experimental evaluation is of course preliminary; many different parameters should be taken into account to compare all approaches. In particular, we plan to study the impact of including wrong terms, either in the content or directory structure, on the query results. Furthermore, we focused this evaluation on photo files, which are content poor. We specifically focused on these to understand the impact of structure in our ranking and ensure that the content dimension scores would not dominate the query evaluation and hide the impact of flexible structural matches. We are currently expanding our experimental evaluation to all types of files.

4 Open Issues

We have presented a framework for flexible query processing over both the content and the structural component in personal information systems. Our results from previous work [11, 12], as well as the ones presented in this paper show that allowing for some approximation in all query dimensions is a promising direction that leads to improvements in the quality of the returned results. Our techniques are a first step in this direction, some open issues still need to be addressed to provide better search capabilities to the user.

- **Integrated Scoring:** Our current scoring approach separately assigns scores to three query dimensions: content, metadata, and structure [12]. Individual dimension scores have an *IDF*-based interpretation and can therefore be combined into a unified score. A more robust approach would be to have a scoring mechanism that integrates all aspects of the query into a single score. Our implementation separates metadata from structure for two reasons: (i) relaxations are defined differently on metadata and structure, and (ii) they are handled separately by the file system. Metadata can be viewed as a structural component of the files and as such integrated in the structural score in a relatively straightforward manner. Integrating content and structure score is a more complex task and requires a better understanding of the interconnections between structure and content. Efforts in the XML community have been made in this direction [9, 5] on a simpler set of structural relaxation rules; we are building upon these to design our integrated scoring.

- **Document vs. files:** The physical representation of a file is not necessarily linked to the logical representation that a user has in mind. For instance, a *LateX* source can be physically divided into multiple files for practical purposes, yet the user logically thinks of all these files as a single document. Conversely, some file types group several logical documents together; this is very common with email files which tend to be a list of individual emails. Flexible query processing should be able to identify these logical boundaries and return the logical document as a result.

- **Granularity of results:** As an extension to the previous point, the query model could consider returning results at different granularity levels in addition to the physical files and logical documents. For instance, photos taken at the same time and location could be returned as a set; or the result of a search comparing job candidates could consist only of the "Education" section of resumes.

Our work shows the importance of allowing for structural query approximation in personal information queries and opens many important open research directions for efficient and high-quality search tools.

5 Acknowledgment

The authors would like to thank Christopher Peery for creating and giving them access to the data set used in this paper.

References

[1] S. Amer-Yahia, N. Koudas, A. Marian, D. Srivastava, and D. Toman. Structure and Content Scoring for XML. In *Proc. of the International Conference on Very Large Databases (VLDB)*, 2005.

[2] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *Proc. of the ACM International Conference on Management of Data (SIGMOD)*, 2004.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. In *Proc. of the ACM International Conference on Management of Data (SIGMOD)*, 2002.

[4] Y. Cai, X. L. Dong, A. Halevy, J. M. Liu, and J. Madhavan. Personal Information Management with SEMEX. In *Proc. of the ACM International Conference on Management of Data (SIGMOD)*, 2005.

[5] D. Carmel, Y. S. Maarek, M. Mandelbrod, Y. Mass, and A. Soffer. Searching XML Documents via XML Fragments. In *Proc. of the ACM International Conference on Research and Development in Information Retrieval (SIGIR)*, 2003.

[6] J.-P. Dittrich and M. A. V. Salles. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *Proc. of the International Conference on Very Large Databases (VLDB)*, 2006.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 2003.

[8] M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: a New Abstraction for Information Management. *SIGMOD Record*, 34(4), 2005.

[9] N. Fuhr and K. Großjohann. XIRQL: An XML Query Language Based on Information Retrieval Concepts. *ACM Transactions on Information Systems (TOIS)*, 22(2), 2004.

[10] Google desktop. `http://desktop.google.com`.

[11] C. Peery, W. Wang, A. Marian, and T. D. Nguyen. Fuzzy Multi-dimensional Search in the Wayfinder File System. In *Proc. of the International Conference on Data Engineering (ICDE)*, 2008.

[12] C. Peery, W. Wang, A. Marian, and T. D. Nguyen. Multi-Dimensional Search for Personal Information Management Systems. In *Proc. of the International Conference on Extending Database Technology (EDBT)*, 2008.

[13] Sleepycat Software. Berkeley DB. http://www.sleepycat.com/.

[14] Apple MAC OS X spotlight. `http://www.apple.com/macosx/features/spotlight`.

[15] An XML Query Language. `http://www.w3.org/TR/xquery/`.

`iMeMex`: From Search to Information Integration and Back

Jens Dittrich — Saarland University
Marcos Antonio Vaz Salles — Cornell University
Lukas Blunschi — ETH Zurich

Abstract

In this paper, we report on lessons learned while building `iMeMex`, the first incarnation of a Personal Dataspace Management System (PDSMS). In contrast to traditional keyword search engines, users may not only search their collections with `iMeMex` but also semantically integrate them over time. As a consequence, the system may improve on precision and recall of queries in a pay-as-you-go fashion. We discuss the impact of several conceptual and architectural decisions made in `iMeMex` and outline open research challenges in combining search and information integration in order to build personal dataspace management systems.

1 Bringing Information Integration to Search in Personal Information

Personal Information Management (PIM) is the process performed daily by users to collect, store, organize, and access their collections of digital objects [3]. Personal information is composed of a heterogeneous data mix, including files, folders, emails, office documents, contacts, images, music, calendar items, videos, among others. In addition to heterogeneity, an added dimension regarding personal information is that it is also distributed, being stored in a wide variety of data sources such as file systems, network shares, external drives, iPods, cell phones, email servers, and on the Web.

There have been two extreme solutions to offer a unified query service over a set of heterogeneous and distributed data sources. At one extreme, traditional search engines allow users to pose simple keyword queries over unstructured sources. XML-IR search engines go one step further and allow users to also pose path queries over semi-structured sources. The semantics of those keyword or path queries are not always precise, meaning that the quality of query answers is evaluated in terms of precision and recall. As a consequence, search engines have difficulties in dealing with queries on structured data, such as grasping the meaning of complex folder hierarchies or finding relationships among documents based on their modification times, authors, or lineage. At the other extreme, an information-integration system allows users to pose complex structural queries over semantically integrated structured or even semi-structured data sources. These systems offer precise query semantics, defined in terms of data models and schemas. However, defining schemas, and mappings among different schemas, is known to be a manual and costly process, which requires the involvement of specialized professionals. As a consequence, information-integration systems are inapplicable to situations in which the number of schemas is too large or users are not specialized, such as in personal information spaces.

In the `iMeMex` project at ETH Zurich, we have explored a new hybrid data management architecture in-between the two extremes of search engines and traditional information-integration systems. We term a system

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

built following this new architecture a *Personal Dataspace Management System (PDSMS)* [2]. The key idea of this new breed of systems is to allow users to smoothly transition from search to information integration. The system should not require any investments in semantic integration before querying services on the data are provided, i.e., at the beginning our system pretty much behaves as a standard search engine. In contrast to search engines, however, our system can be gradually enhanced over time by defining relationships among the data. In that vein, users are able to reach the idiosyncratic level of integration they are most comfortable with for their personal information. The system always strives to offer a unified view of all of the information, but the quality of the integration among items in that view is determined by the amount of integration effort that users are willing to invest over time.

In this paper, we discuss the lessons we have learned while building `iMeMex`, the first incarnation of a PDSMS [1, 2, 4, 5, 6, 13, 22, 23]. We begin by comparing PDSMSs to other common data management architectures in Section 2. In order to illustrate how `iMeMex` mixes search and information integration, we present an example in Section 3. Section 4 proceeds by discussing our choice of data and query model for `iMeMex`. After that, Section 5 discusses advantages and disadvantages of the novel information integration techniques developed to mix search with information integration in `iMeMex`. We discuss system engineering issues in Section 6 and conclude in Section 8.

2 Comparison of PDSMSs to other Systems

In this section, we compare PDSMSs to other data management architectures.

Information-Integration System. Information-integration systems are an all-or-nothing solution: Either schema mappings are available and sources can be queried by the system or these sources are simply not reachable [10, 11, 17, 20, 27]. This limitation hinders their applicability to scenarios in which the number of schemas to integrate is large and the users of the system are not specialized enough to provide schema mappings, as may be the case with personal and social information spaces. In contrast, PDSMSs allow users to query all data sources from the start, albeit with lower precision and recall than if the system knew the relationships among data items. These relationships may be then provided to the PDSMS over time, allowing users to reach the level of result quality and integration most cost-effective for their personal dataspace.

DBMS. With respect to managing personal information, DBMSs have two major limitations: They require a schema to be defined and they take full control of the data [26]. The former limitation is similar to the one faced by information-integration systems. DBMSs are also schema-first systems that present an all-or-nothing solution while PDSMSs allow pay-as-you-go integration of the dataspace. The latter limitation implies that all data must be imported into the DBMS so that the system will provide services over it. As personal dataspaces are a highly heterogeneous and distributed mix, a solution that does not require full control, such as a PDSMS, is more tenable.

Schema-based Dataspace System. Franklin, Halevy, and Maier wrote a vision paper calling for a data co-existence approach to information integration [9]. They termed their envisioned system abstraction a Dataspace Support Platform (DSSP). Although the vision for DSSPs is compelling, little has been shown by Franklin et al. [9] on how to actually achieve it. There have been so far two main research approaches to building DSSPs. The first approach, which we call *schema-based dataspace systems*, takes a traditional information-integration architecture and attempts to relax it in order to reduce manual intervention in the integration process. As an example, probabilistic schema mappings [7] and probabilistic mediated schemas [24] propose that both mappings and mediated schemas be automatically created from the schemas of the data sources. In order to get enough quality, however, mappings must be extracted from a common domain [7, 24]. In addition, this approach still fundamentally relies on declaring schemas and schema mappings before any data source can be integrated into the system. In contrast, PDSMSs offer an alternative approach to building DSSPs. As discussed in the Introduction, PDSMSs are bootstrapped as state-of-the-art search engines, but allow users to increase the level of integration of their dataspace over time, in a pay-as-you-go fashion. In summary, schema-based dataspace systems relax an information-integration architecture to bring it closer to search technology, while PDSMSs power up a search architecture to bring it closer to information-integration technology.

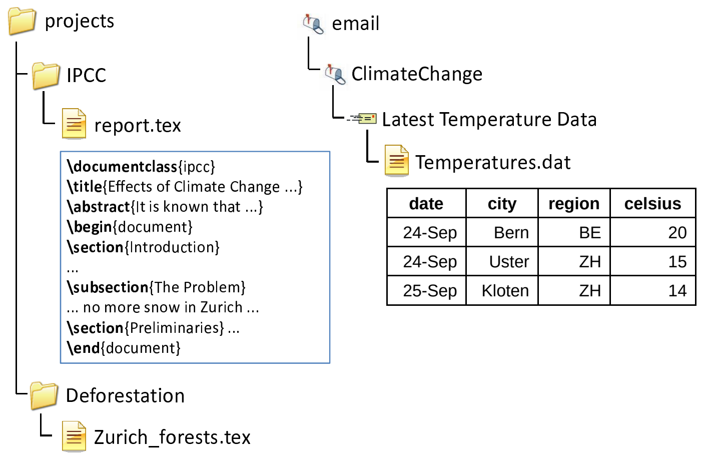
Search Engine. When a PDSMS is bootstrapped, it resembles a state-of-the-art search engine, providing a ranked querying service over a set of heterogeneous sources without any need to declare schemas or mappings. In contrast to a search engine, however, over time, integration hints are added to the PDSMS in order to improve the quality of query results. These hints are *not* full-blown schema mappings, but rather small indications about how the items in the dataspace are related to one another. The system takes advantage of these hints to raise the integration level of the dataspace in a pay-as-you-go fashion.

Cloud Computing Platform. Both PDSMSs and cloud computing platforms are pay-as-you-go solutions, but at different conceptual levels. While a cloud computing platform enables its users to add more hardware and scale incrementally, a PDSMS enables its users to add more relationships and integrate incrementally. There are scenarios in which combining the two characteristics into a common platform is desirable, e.g., for creating an integration system for the whole Web. Further research is necessary to understand how to best bring together these two system abstractions.

Social System. Social systems allow users to establish links to each other and, more importantly, to share information [15]. PDSMSs differ from social systems in that they must enable users to integrate and not only share the information in their dataspace. As such, there must be mechanisms in a PDSMS that allow users to resolve heterogeneity across their data sets and to create arbitrary connections among data items declaratively.

3 Searching Personal Information with `iMeMex`: an Example

As stated earlier, when a PDSMS is bootstrapped, it provides a search service over all data in the sources. Over time, however, users may increase the level of integration of their dataspaces by adding relationships in a pay-as-you-go fashion. In this section, we show an example of how this process works in `iMeMex`.



```
\documentclass{ipcc}
\title{Effects of Climate Change ...}
\abstract{It is known that ...}
\begin{document}
\section{Introduction}
...
\subsection{The Problem}
... no more snow in Zurich ...
\section{Preliminaries} ...
\end{document}
```

date	city	region	celsius
24-Sep	Bern	BE	20
24-Sep	Uster	ZH	15
25-Sep	Kloten	ZH	14

Figure 1: Two data sources in an example personal dataspace.

Suppose a climate-change researcher wishes to search her dataspace for information on the impact of global warming in Zurich. She could start her exploration by posing the keyword query:

```
global warming zurich
```

In the example data sources shown in Figure 1, that simple keyword query returns the files `report.tex` and `Zurich_forests.tex`, appropriately ranked according to term occurrences. There is, however, rich structured data about temperature variations in the region of Zurich hidden in the user's email server. Unfortunately, this data is not returned because the search baseline has no way to translate the keyword query into an appropriate structured query to that data source.

In iMeMex, users may remedy this situation by specifying declarative relationships among portions of their dataspace. These declarative relationships are called *trails*. Users may specify two types of trails: *semantic trails*, which establish relationships among sets of elements, and *association trails*, which establish relationships among individual elements in the dataspace. For example, the user may specify the following semantic trails:

$$T_1 : \texttt{global warming} \longrightarrow \texttt{//projects/IPCC//}\star$$
$$T_2 : \texttt{//projects/IPCC} \longleftrightarrow \texttt{//email/ClimateChange}$$
$$T_3 : \texttt{zurich} \longrightarrow \texttt{region = ``ZH''}$$

T_1 directs the search engine to look for (and rank higher) items under the IPCC filesystem folder whenever the user queries for global warming. T_2 establishes that any search in the context of the IPCC filesystem folder should also consider the information inside the ClimateChange folder in the email server. Finally, T_3 translates keyword searches for zurich into a structural query on the region attribute. Taken together, these *integration hints* enable iMeMex to return the appropriate rows inside the Temperatures.dat file that fulfill the user's information need. In order to achieve that, iMeMex employs query rewrite algorithms that exploit the integration semantics given in the trails [22].

The trails in this example resolve several sources of *heterogeneity* in the dataspace. First, T_1 addresses the gap between the vocabulary employed by users on keyword searches and the hierarchies that users create to organize information on their dataspace. Second, T_2 maps between multiple hierarchies in the dataspace that relate to the same underlying information. Third, T_3 transforms between keywords and specific attribute encodings used in structured sources.

In addition to semantic trails, users may also specify association trails in iMeMex. For example, the user may provide the following association trails to the system:

$$T_4 : \texttt{class=file}\ \theta_1(l,r) \Longleftrightarrow \texttt{class=file},\ \theta_1(l,r) := (\text{r.date}-1 \leq \text{l.date} \leq \text{r.date}+1).$$
$$T_5 : \texttt{class=file}\ \theta_2(l,r) \Longleftrightarrow \texttt{class=email},\ \theta_2(l,r) := (\text{l.author} = \text{r.from}).$$

T_4 states that any two files touched about the same time are related. Thus, when users obtain as a result the report.tex file, for example, the system will also provide as context other files that she was working on when the report was last modified. T_5 relates all files authored by a given person with emails received from that same person. Given this association trail, iMeMex may provide to the user all emails that are related to the report.tex file when it is returned as a query result. Association trails allow users to declaratively specify the *context* in which query results should be placed. Processing association trails requires specific query rewriting and indexing techniques, which we have implemented in iMeMex [23].

4 Data Model and Query Language

We have introduced a new data model to represent personal information in iMeMex called iDM [5]. In a nutshell, iDM is a graph-based model in which nodes contain both attribute-value pairs and unstructured content. The unstructured content and the connections among nodes may be infinite, a feature useful to represent content and data streams. One important aspect of iDM is that it is lazily computed: All nodes and connections in the graph are merely a logical representation of the underlying data in the dataspace. These nodes and connections may be computed dynamically as the graph is navigated or be precomputed for performance reasons. In order to query iDM graphs, we have designed a simple query language, called iQL (iMeMex Query Language) [5, 23]. The core of iQL is composed of keyword and path expressions, in a manner similar in spirit to NEXI [28] and XPath

Full Text [30]. In contrast to those languages, however, iQL semantics are consistent with a graph data model and its syntax is simplified in order to make it more approachable to end users.

The Motivation. A major motivation for developing iDM was the confusion of data model, data, and data representation found in existing models. For instance, in the XML world, a concrete XML snippet like <a><b>hugo</b></a> is actually just a textual data *representation* of some tree-structured *data* conforming to one of the existing XML *data models*. However, from our experience we had seen that most developers assume data representation and data to be equal, i.e. the string "¡a¿¡b¿hugo¡/b¿¡/a¿" is *"XML data"* — which we consider to be incorrect. This confusion has had impact on several unfortunate system design decisions like storing XML data representations, sending XML data representations over the wire, etc. This, however, does not make sense in our view as the data representation is independent of the actual data. This is also one of the reasons why all XML data models are in fact very similar to object models (with some additional syntactic sugar added here and there). The major differentiator in XML is the SGML-like *data representation* of objects.

The Upside. iDM has a crystal clear separation of data and data representation. The flexibility of iDM has been key in allowing us to incorporate all of the heterogeneous data items present in the dataspace into a common model. iDM graphs can uniformly represent data source items such as email messages, files, folders, documents on the Web, social graphs, and database tuples. This also allows us to represent hierarchies available in different formats inside the same logical hierarchy, e.g., the tree-structured content inside a file is just a twig in the larger tree-structured file system. In addition, extracted data is represented in the same model. We have experimented with several different content converters for extraction. Some of them were more general, such as for LaTeX, XML, image metadata, RSS/ATOM streams, or ZIP files, while others were more specific, such as search engine or REST service parsed results.

The Downside. The flexibility that iDM bought us came at the cost of performance and code simplicity. The system had to constantly assume that a general graph was being processed, meaning that clever optimizations developed for tree-structured data were not available to us. In addition, our data model assumed flexible schemas for different nodes in the graph and thus we could not exploit schema information as aggressively as relational implementations. We have tried to reduce the performance hit by having our data source access code export some information about structural properties of the graph being processed. For example, it is useful to detect bridges in the graph as they may be used to reduce memory allocation in graph traversal. While such optimizations brought competitive performance to `iMeMex`, `iMeMex`'s performance remained inferior to what could be obtained by an optimized implementation over simpler data models.

Some dataspace approaches have chosen data models similar to RDF to represent the data in the dataspace [12]. These representations are conceptually simpler than the one introduced by iDM and there has been recent work targeted at providing efficient indexing support [18]. Nevertheless, there are still challenges for a truly efficient implementation when triples represent arbitrary graph patterns and may contain information as disparate as binary file contents and simple integers. In addition, there is little leveraging of schema information, when it is available. We speculate that a more interesting approach for future systems may be to abandon the idea of having a single common data model altogether. This enables the system to use specialized implementations for each specific data model in the dataspace. At the same time, the system must deal with the challenge of providing integration of data in these disparate models without relying on a single common data model but rather using the most appropriate "translation model" at hand.

5 Pay-as-you-go Information Integration

We have developed novel pay-as-you-go information integration techniques that represent the bulk of `iMeMex`'s query planning [22, 23]. As shown in Section 3, the search baseline provided by `iMeMex` may be enriched over time by adding integration hints, called *trails*. We have explored two major classes of trails in `iMeMex`: semantic trails [22] and association trails [23]. Semantic trails model relationships among *sets* of items in a dataspace. A semantic trail can express, for example, that when a user queries for items in the path `//projects/PIM`, then items in the path `//mike/research/PIM` should also be considered. Association trails, on the other hand, model fine-grained relationships among individual *items* in a dataspace. For example, an association trail can express that any document is related to all emails authored by the same person who wrote the document. In other words, semantic trails enrich query results by obtaining sets of semantically equivalent items, while association trails enhance the dataspace by defining connections among individual items.

The Motivation. The major motivation for developing trails was to have a technique that is far easier to use than full-blown information integration, but provides much better query results than standard search engines.

The Upside. In contrast to a full-blown schema mapping, a trail is a simple integration hint that relates two query definitions. There is no need to specify explicit schemas for the sources in the dataspace before a trail can be defined. The queries related by a trail can be any keyword or path query expressed in iQL. As a consequence, trails are at the same time simple enough to enable users to specify them and general enough to capture interesting integration semantics. For example, semantic trails are able to model keyword-to-keyword equivalences as found in thesauri, keyword-to-query bookmarks, or attribute-to-attribute matches as special cases [22]. Association trails define declaratively a graph of connections among instances in the dataspace, being able to relate elements in timelines, through similar content or versions, or by any metadata attached to them [23]. Trails can be shared among different users and cover specialized domains. In addition, we have built in support to attach probability values to trails and to exploit these values to control query rewriting when thousands of uncertain trails are defined in the system.

The Downside. It is clear that candidate trails can be obtained from thesauri [29] and schema matching techniques [21]. It is also clear that relevance feedback can be leveraged in order to ask for confirmations of only the most promising relationships [14]. In spite of that, mining high-quality trails automatically from a user's dataspace remains an elusive goal. We have performed some initial work in that direction for keyword-to-keyword trails [25]. However, a deeper and more interesting question is how to link unstructured, keyword queries to structured, path queries based on the contents of a dataspace and on its usage patterns. This link would allow users to obtain high-quality structured data while only having to interact with a simple keyword query interface.

Another interesting issue we have been faced with is related to the expressiveness of the query language used in the system. Currently, iQL is used both for user queries and for trail definitions. For user queries, a language based on keywords and paths is a natural choice in a personal dataspace scenario. It unifies functionality found in search engines as well as in operating system navigators. However, scripting capabilities such as the ones found in operating system shells could be an interesting addition. For trail definitions, more expressiveness would allow us to define more complex relationships. A negative consequence, however, might be that the query rewriting process will become more complex. As learned from classic information integration, there is a delicate balance to be achieved when deciding on expressiveness, given that fully general queries and relationships may make the query rewriting problem undecidable [16].

6 System Engineering Issues

In order to maintain a system as large as `iMeMex`, the system needs to be separated into components. When we started to implement `iMeMex`, we had a component model in mind and used Java interfaces to connect the different components. Later on we decided to split the code apart into OSGi[1] bundles in order to gain even more flexibility to add and remove components — even at run time. Splitting a large piece of code into OSGi bundles is no easy task and consequently we also only managed to do it partially: The core functionality remained in a central main bundle, the code which was already structured using interfaces was put into smaller bundles. Over

[1]OSGi implementations (e.g. Equinox [8] or Oscar [19]) are platforms for service oriented architectures (SOA). Each service is packaged into a so called OSGi bundle.

time, we chopped away more and more of the core bundle, but still, a part remains.

Motivation. A system such as `iMeMex` offers many opportunities for extensions. More plugins can be added to access different data sources, more data formats may be supported, different indexing mechanisms are useful and query planning and processing can be indefinitely enhanced with additional features. Employing a modular system architecture allows us to add, replace and remove extensions at any point in time.

The Upside. We believe that modularity is key to develop a large system. Working with modules forces developers to follow certain guidelines in terms of what other parts of the system they are allowed to use. Doing so clarifies the system architecture over time and it also allows us to try out new things and still be able to simply get back to an old configuration when we do not observe the intended effect. OSGi naturally enforces such modularity. Modules can also be used as libraries which are shared and distributed among different projects.

The Downside. While OSGi brings modularity, this comes at a cost. Several students complained that adding or changing bundles was too complicated. The reason was that they not only had to write their code, but also had to write a script to package their bundle and configure the system to make use of it. In addition, the strict partitioning into bundles disabled ”quick-hacks”, i.e., adding of new functionality violating bundle boundaries or interfaces. In order to add new functionality, bundle dependencies had to be respected. In the end we had about 70 OSGI bundles. This also shows that we tried to include a lot of functionality into the system. However, this had the cost that some of this functionality was not maintained over time and got ‘lost’ due to disabled tests.

Another downside is that too many levels of abstraction introduce very high function call stacks and it becomes more and more complicated not only to understand what happens, but also to determine where performance goes. One such example that we experienced was the lazy computation of iDM graphs. Accessing a simple tuple attribute could have led to fetching an RSS feed, parsing the XML document and converting it to a large graph of resource nodes — all hidden behind an innocuous `getLastModified()` call.

In the end we took a path in-between modularity and flexibility: The code is still developed in terms of OSGi bundles, but now we also offer a monolithic mode of execution which can be used for testing, debugging and integrating `iMeMex` as a library into other projects.

7 Conclusions

This paper has reviewed several key design choices we took while building `iMeMex`, the first incarnation of a PDSMS. The approach taken in `iMeMex` is to mix search with information integration into a single hybrid platform. We have discussed consequences of our data model and query language, techniques for pay-as-you-go information integration and query processing, as well as system engineering decisions. We believe this experience report may help future designers build the next generation of PDSMSs.

Acknowledgments We would like to thank ETH Zurich, SNF, and in particular Donald Kossmann for making this project possible. We would like to thank all students that contributed to `iMeMex` over the past four years: Tobias Abt, Aarno Aukia, Sandro Blum, Urs Blum, Sibylle Dürr, Markus Färber, Steven Fluck, Pascal Gamper, Olivier Girard, Stefan Hildenbrand, Julia Imhof, Roger Jäggi, Shant Kirakos Karakashian, André Schmidt, Stefan Stalder, Marco Steybe, Philip Stutz, Christian Tarnutzer, and Georg Troxler.

References

[1] L. Blunschi. Cost-based Query Optimization in iMeMex. Master’s thesis, ETH Zurich, 2007.

[2] L. Blunschi, J.-P. Dittrich, O. R. Girard, S. K. Karakashian, and M. A. V. Salles. A Dataspace Odyssey: The iMeMex Personal Dataspace Management System. In *CIDR*, 2007. Demo Paper.

[3] R. Boardman. *Improving Tool Support for Personal Information Management*. PhD thesis, Imperial College London, 2004.

[4] J.-P. Dittrich. iMeMex: A Platform for Personal Dataspace Management. In *SIGIR PIM Workshop*, 2006.

[5] J.-P. Dittrich and M. A. V. Salles. iDM: A Unified and Versatile Data Model for Personal Dataspace Management. In *VLDB*, 2006.

[6] J.-P. Dittrich, M. A. V. Salles, D. Kossmann, and L. Blunschi. iMeMex: Escapes from the Personal Information Jungle. In *VLDB*, 2005. Demo Paper.

[7] X. Dong, A. Halevy, and C. Yu. Data Integration with Uncertainty. In *VLDB*, 2007.

[8] Equinox: Eclipse OSGI implementation. `http://www.eclipse.org/equinox/`.

[9] M. Franklin, A. Halevy, and D. Maier. From Databases to Dataspaces: A New Abstraction for Information Management. *SIGMOD Record*, 34(4):27–33, 2005.

[10] M. Friedman, A. Levy, and T. Millstein. Navigational Plans For Data Integration. In *AAAI*, 1999.

[11] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing Queries across Diverse Data Sources. In *VLDB*, 1997.

[12] B. Howe, D. Maier, and L. Bright. Smoothing the ROI Curve for Scientific Data Management Applications. In *CIDR*, 2007.

[13] iMeMex project web-site. `http://www.imemex.org`.

[14] S. Jeffery, M. Franklin, and A. Halevy. Pay-as-you-go User Feedback for Dataspace Systems. In *SIGMOD*, 2008.

[15] G. Koutrika, B. Bercovitz, R. Ikeda, F. Kaliszan, H. Liou, Z. Zadeh, and H. Garcia-Molina. Social Systems: Can We Do More Than Just Poke Friends? In *CIDR*, 2009.

[16] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, 2002.

[17] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, 1996.

[18] T. Neumann and G. Weikum. RDF-3X: a RISC-style Engine for RDF. *PVLDB*, 1, 2008.

[19] Oscar: OSGi implementation. `http://oscar.objectweb.org/`.

[20] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE*, 1995.

[21] E. Rahm and P. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4), 2001.

[22] M. A. V. Salles, J.-P. Dittrich, S. K. Karakashian, O. R. Girard, and L. Blunschi. iTrails: Pay-as-you-go Information Integration in Dataspaces. In *VLDB*, 2007.

[23] M. V. Salles. *Pay-as-you-go Information Integration in Personal and Social Dataspaces*. PhD thesis, ETH Zurich, 2008.

[24] A. D. Sarma, X. Dong, and A. Halevy. Bootstrapping Pay-As-You-Go Data Integration Systems. In *SIGMOD*, 2008.

[25] A. Schmidt. Semi-Automatic Trail Creation in iMeMex. Master's thesis, ETH Zurich, 2008.

[26] K. A. Shoens, A. Luniewski, P. M. Schwarz, J. W. Stamos, and J. T. II. The Rufus System: Information Organization for Semi-Structured Data. In *VLDB*, 1993.

[27] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer Data Management Systems. In *SIGMOD*, 2004.

[28] A. Trotman and B. Sigurbjrnsson. Narrowed Extended XPath I (NEXI). In *INEX Workshop*, 2004.

[29] WordNet. `http://wordnet.princeton.edu/`.

[30] XQuery and XPath Full Text 1.0, 2008. `http://www.w3.org/TR/xpath-full-text-10/`.

Challenges, Techniques and Directions in Building XSeek: an XML Search Engine

Ziyang Liu Peng Sun Yu Huang Yichuan Cai Yi Chen
Arizona State University
{ziyang.liu, peng.sun, yu.huang.1, yichuan.cai, yi}@asu.edu

Abstract

The importance of supporting keyword searches on XML data has been widely recognized. Different from structured queries, keyword searches are inherently ambiguous due to the inability/unwillingness of users to specify pinpoint semantics. As a result, processing keyword searches involves many unique challenges. In this paper we discuss the motivation, desiderata and challenges in supporting keyword searches on XML data. Then we present an XML keyword search engine, XSeek, which addresses the challenges in several aspects: identifying explicit relevant nodes, identifying implicit relevant nodes, and generating result snippets. At last we discuss the remaining issues and future research directions.

1 Introduction

Information search is an indispensable component of our lives. Due to the vast collections of XML data on the web and in enterprises, providing users with easy access to XML data is highly desirable. The classical way of accessing XML data is through issuing structured queries, such as XPath/XQuery. However, in many applications it is inconvenient or impossible for users to learn these query languages. Besides, the requirement that the user needs to comprehend data schemas may be overwhelming or infeasible, as the schemas are often complex, fast-evolving or unavailable. A natural question to ask is whether we can empower users to effectively access XML data simply using keyword queries.

Unlike text document search where the retrieval unit is an entire document, keyword search on XML data has every XML node as a retrievable unit, thus has a significant potential for fine-grained and high-quality results. Yet it poses a lot of unique challenges of inferring relevant fragments in XML data, composing query results, relevance based ranking, and result presentation. To gracefully process keyword searches on XML, an XML search engine should ideally satisfy a set of desiderata, varying from generating meaningful results to helping users quickly select relevant results. The desiderata include but are not limited to the following ones.

Identifying Explicit Relevant Nodes. A user can specify the required information explicitly using keywords. However, not all nodes matching keywords are necessarily relevant to the query, which need to be distinguished by a search engine. Consider query "*Galleria, Houston*" on Figure 1. For keyword "*Houston*", the match associated with *city* (6) is relevant, as it belongs to a store whose name is Galleria. The *Houston* node associated

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

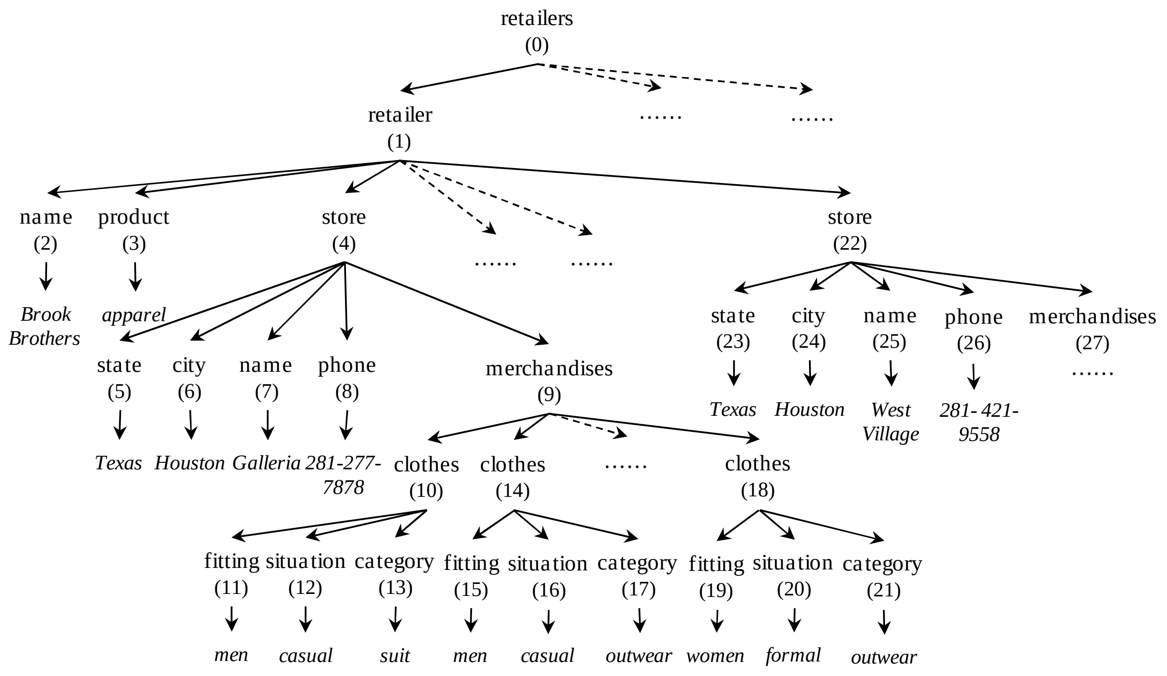


Figure 1: Sample XML Data Fragment

with *city* (24), on the other hand, is irrelevant.

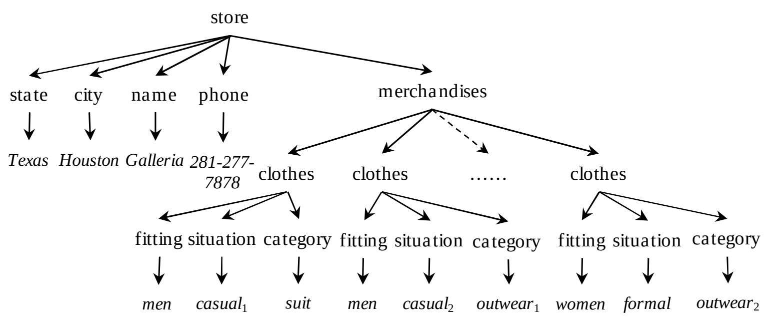
Identifying Implicit Relevant Nodes. Besides the keyword matches that can be relevant, other XML nodes, although do not match keywords, can also be interesting to the user, which are referred to as *implicit relevant nodes*. For the sample query "*Galleria, Houston*", the user is likely interested in the general information of *Galleria* which is located in *Houston*. Therefore, besides returning the keyword matches and their connections, other information of the store, such as its phone number, is also relevant. A result of the sample query is shown as the subtree in Figure 2(a).

Composing Meaningful Results. Given explicit and implicit relevant nodes, the next step is to compose query results using these nodes. Different from text search where a retrieval unit is a document, a result in XML search is a subtree. Given a set of relevant nodes, there can be many ways to separate them into different results, and a meaningful way to do so is desired.

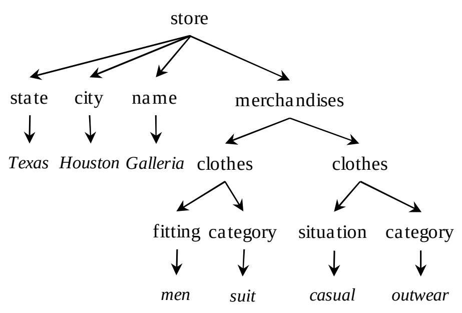
Result Ranking. A keyword search engine usually returns many results, which are not equally relevant to the user. Therefore, it is highly important to rank the results so that the ones that are likely more relevant to the user will be returned earlier.

Generating Result Snippets. While trying to measure the relevance and rank the results, a ranking function is impossible to always correspond to users' preferences. To compensate, result snippets are used by almost every text search engine. The principle of result snippets is to let users quickly judge the relevance of query results by providing a brief quotable passage of each query result, so that users can easily choose and explore relevant ones among many results. For instance, Figure 2(b) is a snippet for the result in Figure 2(a), which highlights the most important information in the result.

In the remaining of this paper we mainly discuss how to achieve three of the desiderata introduced above in building an XML keyword search engine, XSeek [6, 8, 9, 10]: identifying explicit relevant nodes (Section 3.1), identifying implicit relevant nodes (Section 3.2), and generating snippets for query results (Section 3.3). Future directions are discussed in Section 4.



(a) Result Fragments



(b) Snippet of (a)

Figure 2: Result Fragments of Query "*Galleria, Houston*" and its Snippet

2 The *XSeek* System: Architecture

The system architecture of XSeek is presented in Figure 3, which is composed of three main modules: *Index Builder*, *Result Generator* and *Result Presenter*. As their names indicate, *Index Builder* builds data indexes which speed up query processing; *Result Generator* takes query keywords as input and generates results; *Result Presenter* presents the results to the user in such a way that the users can quickly find relevant results. The modules are described in detail in the following.

Index Builder. The *Index Builder* parses the input XML data, infers the inherent entities and attributes (to be explained in Section 3) in the data, and builds indexes for retrieving the information about node category, parent and children.

Result Generator. Once a user issues a query, *Explicit Relevant Node Identifier* accesses the indexes, retrieves matches to each keyword, and identifies the matches that are relevant to the query. Then, *Implicit Relevant Node Identifier* identifies other relevant nodes besides keyword matches. Specifically, *Keyword Categorizer* analyzes the match patterns and categorizes input keywords as predicate nodes and return nodes. Based on the analysis of keyword match patterns and entities in the data, *Return Node Identifier* recognizes return nodes for the query. *Query Result Composer* generates query results based on the relevant nodes identified in the previous steps.

Result Presenter. *Query Result Ranker* ranks the results generated by *Query Result Composer* based on diverse ranking factors, such as variants of Term Frequency (TF), Inverse Document Frequency (IDF), PageRank, the number of keyword matches, proximity of keyword matches, query result size, etc.. However, instead of having all the results generated and then ranked, *Query Result Composer* and *Query Result Ranker* work interactively in order to generate query results in an (approximate) ranked order, so that the top-ranked results can be returned in a short time. *Result Snippet Generator* provides a self-contained, distinguishable, representative and concise summary for each query result to help users quickly judge the relevance of results.

3 Important Modules in XSeek

In this section, we discuss XSeek modules that address several challenges of XML search. We model XML data as a rooted, labeled and unordered tree, such as the one shown in Figure 1. Every internal node in the tree has a node name, and every leaf node has a data value. A keyword search is a query consisting of a set of keywords, and a result of a keyword search is a subtree which consists of selected nodes in the XML tree along with their connecting paths. XSeek adopts AND semantics, i.e., each result contains all keywords in the query.

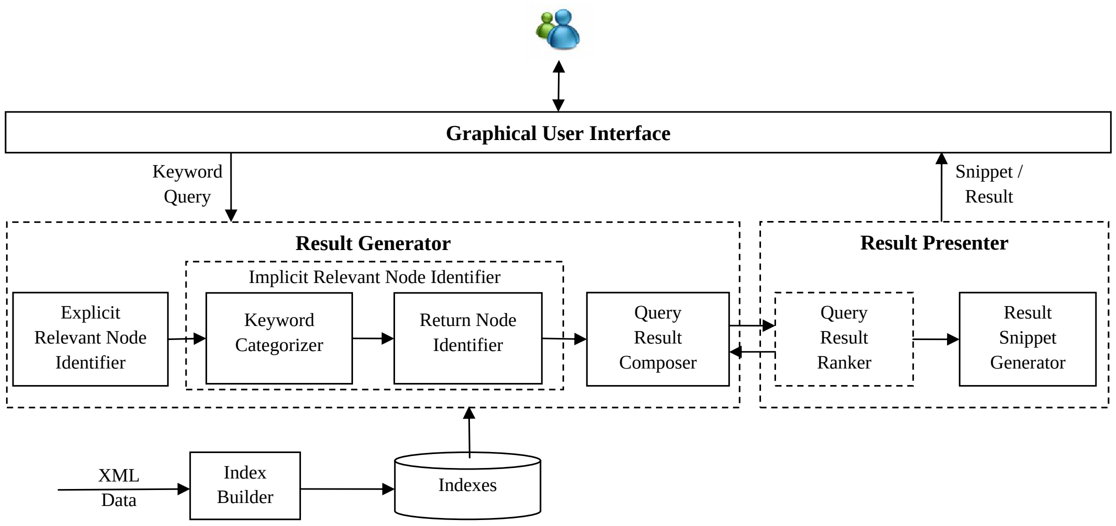


Figure 3: Architecture of XSeek

3.1 Explicit Relevant Node Identifier

As discussed in Section 1, explicit relevant nodes are matches to query keywords which are deemed relevant. To identify explicit relevant nodes, XSeek begins with computing the *smallest lowest common ancestor* (SLCA) nodes [12] of a keyword search. The SLCAs of a query Q consist of nodes which contain all keywords in their subtrees, and none of their descendants contains all keywords in its subtree. Computing SLCA helps prune some irrelevant matches. For example, the only SLCA node of query "*Galleria, Houston*" on Figure 1 is node *store* (4), thus the *Houston* node associated with *city* (24) is pruned. However, not all keyword matches in the subtrees of SLCAs are relevant, e.g., for query "*Brook Brothers, Galleria, Houston*", the SLCA is *retailer* (1), however, the *Houston* node associated with *city* (24) is still irrelevant.

XSeek further prunes irrelevant matches by pruning inferior subtrees. A subtree is considered inferior and is pruned, if the root of the subtree has a sibling whose subtree contains strictly more keywords. Consider query "*Brook Brothers, Galleria, Houston*" again. For node *store* (4), the set of keywords contained in its subtree is $\{$*Galleria, Houston*$\}$. Assume that there is no *Galleria* node in the subtree rooted at *store* (22), i.e., it only contains keywords $\{$*Houston*$\}$. As a result, node *store* (22) is pruned along with its subtree, thus the *Houston* node in this subtree is not considered as an explicit relevant node.

There are many other strategies for identifying explicit relevant nodes, and a question is how to evaluate their effectiveness. The effectiveness of a keyword search approach is usually evaluated based on the ground truth obtained from user study. We proposed a cost-effective evaluation alternative [9], which uses an axiomatic framework consisting of the following four desirable properties for identifying explicit relevant nodes: data/query monotonicity/consistency. These properties capture the reasonable behaviors of an XML search engine adopting AND semantics upon a change to the data or the query.

Data Monotonicity. If we add a new node to the data, then the data content becomes richer, therefore the number of query results should be (non-strictly) monotonically increasing. Analogous to keyword search on text documents, adding a word to a document that is not originally a query result may qualify the document as a new query result.

Query Monotonicity. If we add a keyword to the query, then the query becomes more restrictive, therefore the number of query results should be (non-strictly) monotonically decreasing. Analogous to keyword search on text documents, adding a keyword to the query can disqualify a document that is originally a query result to be a result of the new query.

Data Consistency. After a data insertion, each additional subtree that becomes (part of) a query result should contain the newly inserted node. Analogous to keyword search on text documents, after we add a new word to the data, if there is a document that becomes a new query result, then this document must contain the newly inserted word.

Query Consistency. If we add a new keyword to the query, then each additional subtree that becomes (part of) a query result should contain at least one match to this query keyword. Analogous to keyword search on text documents, after we add a new keyword to the query, if a document remains to be a query result, then it must contain a match to the new keyword.

As an example of data monotonicity, consider query "*Galleria, Houston*" issued on Figure 1. Suppose a search engine outputs a set of results, each being a subtree rooted at a *store* node that contains *Galleria* and *Houston* as descendants. Now suppose that we add one more *store* node to the data that has *Galleria* and *Houston* as descendants. If the same search engine outputs fewer results after the data insertion, then it is counter-intuitive, and violates data monotonicity.

As an example of query consistency, consider query "*Galleria, Houston*" again. For this query, suppose a search engine outputs a set of results, each rooted at a *store* node that has *Galleria* and *Houston* in its subtrees; the *Houston* node associated with *city* (24) is considered as irrelevant. Now if we add a new query keyword *Brook Brothers*, and the query becomes "*Brook Brothers, Galleria, Houston*". If the same XML search engine considers the *Houston* node associated with *city* (24) as relevant for the second query, it violates query consistency. Indeed, this is unlikely to be desirable.

Although these properties appear to be intuitive and simple, it is not trivial to satisfy them. It was shown that XSeek is the only known approach that satisfies all four properties [9].

3.2 Implicit Relevant Node Identifier

Besides relevant keyword matches identified by the *Explicit Relevant Node Identifier*, other XML nodes that do not match keywords may also be relevant, which are referred to as *implicit relevant nodes*, and are identified by the *Implicit Relevant Node Identifier* module.

Consider two queries Q_1: "*Galleria, Houston*", and Q_2: "*Galleria, city*". By issuing Q_1, the user is likely interested in information about a store whose name is Galleria and which is located in Houston. In contrast, Q_2 indicates that the user is likely interested in a particular piece of information: the city of Galleria. Note that if we only consider explicit relevant nodes, then these two queries have similar results. However, they have different semantics and their results should be different. This can be addressed by considering implicit relevant nodes. The implicit relevant nodes for Q_1, intuitively, include those in the subtree rooted at *store* (4), while the implicit relevant nodes for Q_2 include those in the subtree rooted at *city* (6).

To infer implicit relevant nodes, we can obtain some hints by analyzing keyword categories. From these two queries, we can see that an input keyword can specify a *predicate* that restricts the search (analogous to "where" clause in XQuery), such as *Galleria* and *Houston* in Q_1. Or, a keyword may specify a desired return node (analogous to "return" clause in XQuery), such as *city* in Q_2. The subtree rooted at a return node, albeit not necessarily on the paths connecting relevant keyword matches, is usually interesting to the user. For Q_2, for instance, the return node is *city* (6), and therefore, its child node *Houston* is an implicit relevant node.

However, return nodes are not always specified in a keyword query, such as Q_1 in which both keywords specify predicates. In this case, we believe that the user is interested in the general information about the data entities related to the search, and consider such entities as return nodes. For instance, in Q_1 the user is likely to be interested in the general information about entity *store* (4), and thus *store* is considered as the return node for this query.

XSeek uses the following heuristics to infer keyword match patterns and data semantics to infer return nodes.

Analyzing Keyword Match Pattern. The *Keyword Categorizer* module of XSeek infers keyword categories using the following rules:

1. If a keyword k_1 has a match u which is a name node (i.e., element or attribute name), and there does not exist an input keyword k_2 with match v, such that u is an ancestor of v, then we consider k_1 as a return node.

2. A keyword that is not a return node is treated as a predicate.

Indeed, if a keyword matches a name node but none of its descendants are specified in the query, then very likely the user is interested in the detailed information of this node, which appears in its subtree. For example, keyword *city* in Q_2 is a return node as it matches name node (6) with no other keyword matching its child, *Houston*. This indicates that the user is likely interested in the information of *city*, which is *Houston*. In Q_1, *Houston* is considered as a predicate since its match is a value node.

Analyzing Data Semantics. In the case that return nodes are not specified in the keywords, XSeek infers return nodes by analyzing XML data semantics. Specifically, the *Return Node Identifier* module categorizes XML data nodes into three categories: entity, attribute and connection node, using the following rules:

1. A node represents an *entity* if it corresponds to a *-node in the DTD.

2. A node, together with its child, denotes an *attribute* if it does not correspond to a *-node, and only has one leaf child.

3. A node is a *connection node* if it represents neither an entity nor an attribute.

When there is no return node specified in the query, XSeek identifies return nodes as the entity nodes that are in the subtrees rooted at SLCAs.

Besides outputting the paths connecting each SLCA and relevant keyword matches, XSeek also outputs the subtree rooted at each return node to make the result meaningful. The result of Q_1 "*Galleria, Houston*", in which both keywords are inferred as predicates and entities *store* and *clothes* are considered as return nodes, is presented in Figure 2(a).

3.3 Result Snippet Generator

XSeek is the first XML search engine which generates snippets for query results. Compared with result snippets for text document search, XML presents more opportunities for generating meaningful result snippets due to its semi-structured nature with mark-ups providing meaningful annotations to data content.

Continue with the sample query "*Galleria, Houston*" and the result shown in Figure 2(a). To generate result snippets, XSeek selects *informative items* from the result into *a snippet information list*, which is initialized with the list of keywords, such as the items in step 1 in Figure 4. Then, it selects instances of the items in the list from a result to construct a snippet. The items in the information list are selected from the result based on the following goals: self-contained, distinguishable, representative, and small.

Self-Contained. A result snippet should be *self-contained* so that the users can understand it. In text search, result snippets usually consist of one or more "windows" on the document containing the complete phrases/sentences where the keyword matches appear, which are self-contained and can be easily read. In contrast, in XML documents, each entity (Section 3.2) is a semantic unit. XSeek includes the names of the entities involved in the query result as context information, even though such entity names may not necessarily appear between two keyword matches in the XML document. For query "*Galleria, Houston*", entities *store* and *clothes* are included in the snippet information list, as shown in step 2 in Figure 4.

Distinguishable. Different result snippets should be *distinguishable* so that the users can differentiate the results from their snippets with little effort. To achieve this in text document search, result snippets often include the document titles. In XML keyword search, intuitively, the key of a result can be used for distinguishing results. A query result may contain several entities, thus the question is the keys of which entities should be considered as the key of the query result. XSeek classifies the entities in a query result into two categories.

1. return entities are what the users are looking for when issuing the query.

2. supporting entities are used to describe the return entities in a query result.

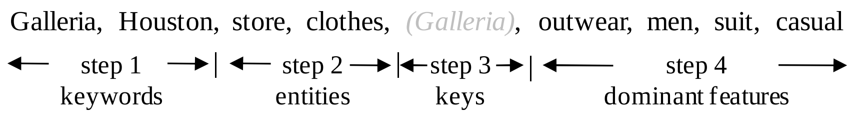

Galleria, Houston, store, clothes, (Galleria), outwear, men, suit, casual
step 1
keywords
step 2
entities
step 3
keys
step 4
dominant features

Figure 4: Snippet Information List

Since return entities reflect the search goal, XSeek considers the keys of return entities as the keys of a result. XSeek adopts the following heuristics for identifying return entities: an entity in a query result is considered a *return entity* if it is associated with a return node (i.e., it is itself a return node, or it has an attribute which is a return node). If there are multiple such entities, then the ones with biggest heights are considered as the return entities.

The key of XML nodes can be directly obtained from the schema of XML data (if available), specified as ID or Key node. Otherwise, XSeek finds the most selective attribute of the return entity and uses it as the key. For query "*Galleria, Houston*", the return node is *store*, therefore *store* is considered as the return entity. Its key, *Galleria*, is included in the snippet information list (step 3 in Figure 4).[1]

Representative. A snippet should be *representative* to the query result, thus the users can grasp the essence of the result from its snippet. Intuitively, the most prominent features in the result should be shown in its corresponding snippet. A feature is defined as a triplet (entity name e, attribute name a, attribute value v). For example *clothes:fitting:men* is a feature. The pair (entity name e, attribute name a) is referred as the type of a feature, and attribute value v is referred as the value of a feature.

Intuitively, a feature is considered *dominant* if it has a large number of occurrences compared with features of the same type, quantified by its dominance score. For example, if most clothes in Figure 2(a) are for *men*, then feature *clothes:fitting:men* is a dominant feature. Features are added to the snippet information list in the order of their dominance score, as shown in the last step of Figure 4.

Small. A result snippet should be *small* so that the users can quickly browse several snippets. XSeek generates snippets subject to an upper bound of snippet size (in terms of the number of nodes), which can be specified by users.

An item in the snippet information list can have multiple occurrences in the query result. Although different instances of the same informative item are not distinguishable in terms of semantics, they have different impacts on the size of the snippet. For example, in Figure 2(a), to include two items *casual* and *outwear*, choosing $casual_2$ and $outwear_1$ results in a smaller tree than choosing $casual_1$ and $outwear_1$.

The problem of maximizing the number of informative items selected in a snippet given the snippet size upper bound is hard. The decision version of this problem is proved to be NP-complete [6]. XSeek uses a greedy algorithm that can efficiently select instances of informative items in generating a meaningful and informative snippet for each query result given an upper bound on size. The snippet of Figure 2(a) is shown in Figure 2(b).

4 Conclusions and Future Research Directions

We have introduced the desiderata and challenges in developing XML keyword search engines, and presented how to address some of them in XSeek. In this section we summarize the remaining issues in building a full-fledged XML search engine and future research directions.

Ranking and Top-k Query Processing. Due to the inherent ambiguity of keyword searches, ranking is highly important for any keyword search engine. A ranking scheme considers both ranking factors for individual nodes, such as variants of TF, IDF, PageRank, and ranking factors for the entire result, such as the aggregation of individual node scores, keyword proximity and result size [3, 4, 5, 7].

[1]Item *Galleria* is already included in the snippet information list in step 1.

However, none of the existing approaches on XML keyword search supports top-k query processing. They need to generate all the results, compute a score for each result and finally sort them. This can be potentially very inefficient when the number of results is large. It is highly desirable but challenging to develop efficient top-k algorithms for XML keyword search, which, through the interaction of *Query Result Composer* and *Query Result Ranker* modules in Figure 3, generates top-k query results without producing all results.

Utilization of User Relevance Feedbacks. In information retrieval, user relevance feedbacks have been widely exploited for deducing users' interests and have achieved big success in improving search quality. Existing work on utilizing feedbacks for XML search focuses on finding the structural relationships among keywords using explicit feedbacks [11]. How to use relevance feedbacks, especially implicit feedbacks, in different aspects of XML keyword search is largely an open question. With the structure of XML data, there are plenty of opportunities of utilizing user relevance feedbacks to improve search quality, from relevant node inference in result generation to ranking and snippet generation in result presentation in a keyword search engine.

Evaluation of XML Keyword Search Systems. Since there are many approaches of generating and presenting results for XML keyword search, a framework to evaluate them is critical for users to understand the trade-offs of existing systems. The quality of an XML keyword search system is often gauged empirically with respect to the ground truth over a large and comprehensive set of test data and queries. INEX [1] is an initiative for developing empirical benchmark for evaluating XML search. Two important factors are proposed to measure result relevance: exhaustivity, measuring how well a node satisfies user's information need; and specificity, measuring how well a node focuses on the user's information need [2]. A cost-effective alternative to an empirical benchmark is to evaluate XML keyword search systems based on a set of axioms that capture broad intuitions, such as [9] which evaluates strategies for identifying relevant keyword matches using axioms (as reviewed in Section 3.1). We believe it is essential for the community to actively contribute to comprehensive frameworks for XML keyword search evaluation.

5 Acknowledgement

This material is based on work partially supported by NSF CAREER award IIS-0845647 and NSF grant IIS-0740129.

References

[1] INitiative for the Evaluation of XML Retrieval. http://inex.is.informatik.uni-duisburg.de/.

[2] S. Amer-Yahia and M. Lalmas. XML Search: Languages, INEX and Scoring. *SIGMOD Rec.*, 35(4), 2006.

[3] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective XML Keyword Search with Relevance Oriented Ranking. In *ICDE*, 2009.

[4] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML, 2003.

[5] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD*, 2007.

[6] Y. Huang, Z. Liu, and Y. Chen. Query Biased Snippet Generation in XML Search. In *SIGMOD*, 2008.

[7] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: Efficient and Adaptive Keyword Search on Unstructured, Semi-structured and Structured Data. In *SIGMOD*, 2008.

[8] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In *SIGMOD*, 2007.

[9] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *VLDB*, 2008.

[10] Z. Liu and Y. Chen. Answering Keyword Queries on XML Using Materialized Views. In *ICDE*, 2008 (Poster).

[11] R. Schenkel and M. Theobald. Structural Feedback for Keyword-Based XML Retrieval. In *ECIR*, 2006.

[12] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.

Searching Shared Content in Communities with the Data Ring

Serge Abiteboul[*]
INRIA-Saclay
Serge.Abiteboul@inria.fr

Neoklis Polyzotis[†]
Univ. of California Santa Cruz
alkis@ucsc.edu

Abstract

Information ubiquity has created a large crowd of users (most notably scientists), who could employ DBMS technology to share and search their data more effectively. Still, this user base prefers to keep its data in files that can be easily managed by applications such as spreadsheets, rather than deal with the complexity and rigidity of modern database systems.

In this article, we describe a vision for enabling non-experts, such as scientists, to build content sharing communities *in a true database fashion: declaratively. The proposed infrastructure, called the* data ring*, enables users to share and search their data with minimal effort; the user points to the data that should be shared, and the data ring becomes responsible for automatically indexing the data (to make it accessible), replicating it (for availability), and reorganizing its physical storage (for better query performance). We outline the salient features of our proposal, and outline recent technical advancements in realizing data rings.*

1 Introduction

Imagine a community of scientists who collect experimental data. It is common within such communities to share data in order to promote more research and enable the quick dissemination of information. (SWISSPROT[1] and the Genome Browser[2] are characteristic examples of this scenario.) In turn, content sharing necessitates a service for searching and/or querying, to enable the discovery of useful information out of the collected data. These two elements of sharing and searching can be found in other contexts as well, especially with the Web and the creation of user communities, such as Flickr. This evidence indicates an emerging trend towards building what we call *content sharing communities*: groups of users that wish to share and query information in some specific domain.

In principle, a distributed DBMS provides all the services needed to support content sharing communities. In practice, however, users avoid using DBMS technology due to its complexity. For instance, the conceptually simple operation of loading data in a DBMS involves several tasks, such as, defining a schema, tuning the

[*]S. Abiteboul is a member of LRI, University Paris Sud and LSV, ENS Cachan. His work was partially supported by the ERC Grant Webdam and the ANR Grant Dataring.

[†]Part of the work was performed while the author was visiting INRIA, France.

[1]`http://www.expasy.org/sprot/`

[2]`http://genome.ucsc.edu`

database, collecting statistics, etc., that are not trivial for non-experts. These tasks are dwarfed in complexity by the next logical step, which is linking the local databases in a distributed or federated system. And even if a database expert is found to perform all these time consuming tasks, users may still find a conventional database system too rigid when it comes to its control over the data. Essentially, data has to be imported in the DBMS before it can be indexed and searched, and it has to be exported before it can be processed by other software.

We believe that the aforementioned user base forms a formidable challenge for database researchers and a chance to bring database systems to the "masses". In this direction, we propose the *data ring* that can be seen as a network analogue of a database or a content warehouse. The vision is to build a P2P middleware system that can be used by a community of non-experts, such as scientists, to build content sharing communities in a declarative fashion. Essentially, a peer joins a data ring by designating which files (or services in general) are to be shared, without having to specify a schema for the data, load it in a store, create any indices on it, or specify anything complex regarding its distribution. The data ring enables users to perform searches or declarative queries over the aggregated data, and becomes responsible for reorganizing the physical storage of data and for controlling its distribution. Thus a primary focus of the data ring is *simplicity of use*. The inherent diversity and complexity is hidden behind a simple and unified interface for publishing, querying, monitoring, integrating, and updating the available information. Besides clicking to select resources, the use of a ring resource should be no more complicated than the use of the same resource in a familiar local system.

As previously stressed, data rings target non-expert users, implying that the deployment of the ring and its administration should be almost effort-less. The "pay-as-you-go" philosophy (promoted by "dataspaces" [11]) is replaced here by "pay-only-by-providing-content"; the physical organization is (almost) for free. A first essential facet of the data ring proposal is thus the use of fully automatic and distributed data administration, both for data residing in file systems (a most common situation) and for more controlled systems, such as databases. Access structures, e.g., materialized views or indexes, are automatically introduced by the system when needed. Also, the system may choose to "activate" static data residing in the file system, e.g. a large collection of RSS seeds, to assign its management to a more efficient system such as a relational DBMS. The relational system is then in charge of managing the collection and providing efficient mechanisms for querying and monitoring.

The remainder of the article briefly reviews the salient features of the data ring proposal, insisting on its most novel and challenging aspects. We also discuss recent technical advancements in realizing a distributed data catalog that supports the search functionality of the data ring system. For both parts, the complete details can be found in the respective publications [3, 4].

2 Related Work

In principle, a data sharing community can be maintained as a distributed database using existing systems. Under this model, each user runs a local DBMS on the data that he/she wishes to publish, and all the participating databases are connected in a loose federation that forms a virtual global repository. As mentioned earlier, the main issue of this approach is the increased complexity of setting up and tuning such a system. It should be noted that modern database systems are equipped with several "advisors" (e.g., [5]) that can assist users in the difficult task of system maintenance. Such tools, however, implicitly assume that the user has some experience with database systems and can thus make informed decisions based on the recommendations that the advisors generate. Moreover, the proposed techniques focus primarily on the tuning of a centralized system, and it is not clear if they can be extended to the equally important problem of tuning the distributed system.

Peer-to-Peer (P2P) file sharing systems, such as, Gnutella or eDonkey, offer a light-weight alternative to the complexity of a full blown database system. Their main shortcoming is that they allow the location of files based solely on name matching, whereas users in data sharing communities are primarily interested in locating data based on content. Moreover, P2P file-sharing systems typically rely on query flooding as the main query processing mechanism, which does not scale well for complex queries and high querying rates.

Several research projects in academia have investigated the development of P2P database systems as the platform for supporting data sharing communities. In particular, previous works have investigated issues related to data integration [20], system design [9], and processing of complex queries [15, 12] over P2P networks. These aspects are undoubtedly important in the development of tools for the creation of data sharing communities. As noted earlier, however, an important issue is the development of mechanisms for self-administration that allow the P2P system to operate efficiently in an autonomous fashion. In this direction, earlier studies [13, 10] have investigated self-tuning techniques for the DHT [19, 18] substrate of P2P systems. These works are clearly an important step in the realization of self-managed distributed systems. It is equally important, of course, to explore self-tuning mechanisms for the upper layers of the system, that provide the support for complex queries over the distributed data.

The data ring proposal borrows ideas from several previous works on distributed data management. From P2P content warehouse [1], we adopt the idea of using semantic tools to enrich the data at our disposal, e.g. by discovering links between pieces of data. From [11], we borrow the vision of a *dataspace* where data is integrated. Optimization techniques can be used from the existing large body of works on distributed query optimization, such as PIER [15] and ActiveXML [6]. In particular, our proposal resembles the goals of PIER [15] in that it introduces a middleware system for running distributed query processing applications. The main difference is that we focus on a system to be used by non-experts, insisting on declarative management wherever possible.

3 The Data Ring: Design Overview

At an abstract level, a data ring is formed by a collection of peers, where each peer exports a set of resources (services and data). The collaborating peers are autonomous, heterogeneous, and their capabilities may greatly vary, e.g., from a sensor to a large database. Moreover, their exported resources may be quite diverse, e.g., some peers enable access to data resources, such as, a phone book or a genome bank, while others offer computational resources, such as, an ontology-based classifier, or a gene matching library. In what follows, we use P to denote a peer in the data ring, and $r@P$ to denote a resource r that peer P exports. A resource r can be either a data item or a service that P hosts.

We advocate a data model based on ActiveXML in order to represent services and data in a unified framework. More specifically, we assume that each data item is represented with an XML view, while services are specified in the Web Services Description Language [8] (WSDL). Two observations are in order. First, we employ XML solely as a model for data integration, and do not insist that each peer uses XML as its native data model. We assume that the latter is determined on a per-peer basis, depending on the type of exported data. Second, we advocate an enriched XML model that can capture both extensional and intensional data, i.e., data obtained through web service calls.

Peers can query the shared information using tree-pattern queries. We choose this formalism since it forms the basis of the structured FLWR construct of XQuery, and also because it is flexible enough to express unstructured search queries based on keywords and element tags. Figure 1 shows two example tree-pattern queries corresponding to these cases. A tree-pattern query issued at some peer is forwarded, though the data ring, to peers containing relevant data, where it is translated locally to a native query over the published resources. Therefore, the data ring borrows several features from mediation systems [16].

Following the established principles of database systems, a data ring organizes information at three levels of abstraction: the external layer, the topological layer, and the physical layer. (See Figure 2). The following summarizes the functionality of each part.

Topological Layer The topological layer contains the logical description of the information managed by the data ring, i.e., the set of participating peers and the resources that they export. While the topological

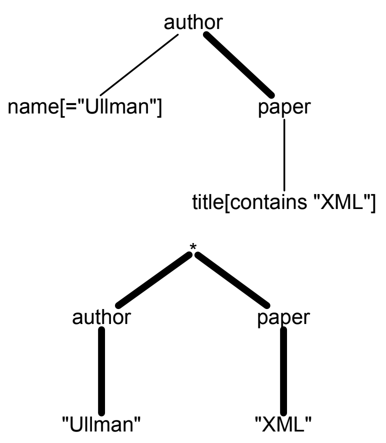


Figure 1: Two example tree-pattern queries over a repository of AXML documents with bibliographic publications. The top query applies specific structural constraints on the matched elements, whereas the bottom query only specifies containment predicates without restricting the structure.

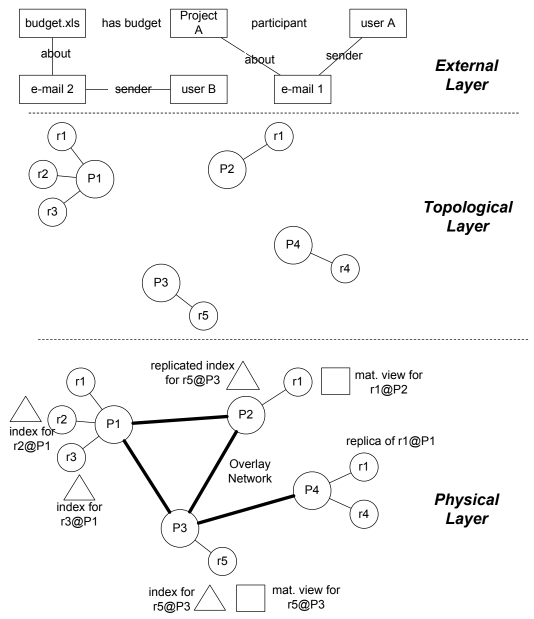


Figure 2: Information abstraction in a data ring.

layer does not expose the overlay network that is used to realize the distributed system, it does include information on the location[3] of resources, i.e., the $@P$ specification.

The topological layer provides declarative query services, in the same way that the logical model of a relational database supports SQL queries. A peer can thus submit an ad-hoc query against the registered resources, monitor a resource for changes, or open a continuous query against a stream resource (e.g., sensor readings, or an RSS stream). We also assume that the topological layer records the lineage and uncertainty properties of the data, and supports their integration in the query language.

External Layer As described previously, the topological layer manages different kinds of information: traditional data (as in relational systems), content (mails, letters, reports, etc.), metadata, as well as domain specific knowledge (e.g., ontologies) needed to interpret data and metadata. Since this level of detail can still be daunting for a large class of users, we enable the generation and maintenance of semantically richer data models in the *external layer*. For instance, the external layer can house the data models that are proposed in dataspaces [11]. In this fashion, a user will observe concepts and relationships between concepts, and will pose semantically rich search queries such as “what is the name of John Doe’s company”, typically using some simple syntax or some forms-based interface. We expect such queries to be realized by combining declarative queries over the topological layer, driven by the semantic information in the external data model.

Physical Layer The physical layer supports the evaluation of distributed query plans over the registered resources. It comprises a physical model and language for distributed query evaluation, as well as physical structures for managing the registered data. In particular, the physical layer provides a DHT service through some overlay network [19], on top of which the data ring maintains a catalog of meta data on the published resources. The meta-data includes the location of resources, information on their con-

[3]This is why we use “topological” instead of the more common term “logical” for the identification of the layer.

tents/capabilities, statistics on data and workload distribution, and in general any information that is pertinent to the evaluation of distributed queries. In general, we distinguish two key features of the physical layer: (a) it is completely distributed, and (b) it is self-administrated. These two elements combined introduce novel challenges in the design of autonomic systems.

Our work in the data ring project focuses mostly on the topological and physical layers, which we discuss further in the coming sections. This is not to say that we find the external layer to be less important; on the contrary! We hope to find some synergy here with works in dataspace support platforms [11], which examine data models similar to the ones that we envision for the external layer.

4 Distributed Catalog

As mentioned earlier, the physical layer maintains a catalog of meta data for the published content. Among other services, the catalog provides the following functionality that is relevant for our discussion: Given a tree-pattern query Q, it returns a super-set of peers that publish content relevant to Q. The idea is that a peer searches for content by first consulting the catalog with the corresponding query Q and then forwarding Q to the returned peers. Accordingly, a peer publishes resources in the data ring by pushing the corresponding meta data to the catalog. Overall, the catalog can be viewed as a distributed index of the published content.

For the purpose of indexing, each element in a published AXML document is identified by a structural identifier $sid = (start, end, lev)$. Here, $start$ (resp. end) is the number assigned to the opening (resp. closing) tag of the element, when reading the document and numbering its tags in the order they appear in the document. The third value lev denotes the element's level in the tree. Structural ids allow deciding if element e_1 is an ancestor of element e_2 by verifying if $e_1.start<e_2.start<e_1.end$. If $e_1.lev = e_2.lev$ holds in addition, then e_1 is the parent of e_2.

The catalog indexes element labels as well as words[4] in documents. We henceforth use *term* to refer to either of the two. The indexing scheme is based on the *Term* relation, defined as follows:

Term(p,d,sid,l) l is the label of element (p, d, sid)
Term(p,d,sid,w) w is a word under element (p, d, sid)

We refer to a tuple in *Term* as a *posting*. Given a term a, we refer to the set of its postings as the *posting list* for a and denote it as L_a.

In the data ring, XML documents are stored at their publishing peer, whereas the *Term* relation is distributed among the peers of the system using the DHT service of the physical layer [19, 18]. The keys of the DHT entries are the term values, and thus each peer becomes in charge of a set of posting lists. We use *Term*$_p$ to denote the horizontal fragment of *Term* that is stored in peer p. Given a query Q, the data ring retrieves the posting list L_a for each tag a that appears in Q and performs a parallel holistic twig join [7] to identify matching elements. The structural ids of the matching elements point to the peers to which Q is forwarded for further processing.

We developed two optimizations in order to improve the efficiency of the catalog, namely, distributed posting partitioning and structural bloom filters. Distributed posting partitioning (DPP for short) addresses the performance problems resulting from the high skew in the distribution of posting list sizes. The idea is to split the posting list for a popular term horizontally in *blocks* based on range conditions on structural identifiers, and to migrate portions to other peers. This leads to a hierarchical organization of the posting lists in the style of distributed B-trees [17]. Thus, a request for the posting list L_a is satisfied by doing parallel transfers for the blocks of L_a, which leads to higher efficiency. The holistic twig join is also modified to take into account the new organization. Specifically, the algorithm examines the range conditions that define the split of L_a in order to prune blocks that provably do not join with blocks from other lists.

[4]For efficiency, stop words are excluded from the index.

The second optimization of *Structural Bloom Filters* (SBFs for short) is orthogonal to DPP and aims to reduce the total volume of transferred data for the twig join. A SBF provides a compact representation of a set of postings that is suitable for filtering the postings of another list. As an example, assume that the join operator at the query peer P checks the descendant condition $a//b$ between the posting lists L_a and L_b located at peers P_a and P_b respectively. SBFs enable the following strategy for avoiding the transfer of the complete lists: P_a computes locally a small SBF F_a that summarizes L_a; P_b receives F_a and uses it to select a sublist $L'_a \subseteq L_a$ that provably contains all the b postings that are descendants of a; finally, the join operator at P performs the join between L_a and L'_b. Thus, the idea is to perform some local computation and exchange a small filter in order to avoid sending b postings that are provably not in the result of the twig join. Similar to bloom filters for relational data, SBFs commit only false positive errors with a bounded probability that can be tuned. We also develop several query processing strategies that utilize SBFs and that offer different trade-offs depending on the characteristics of the tree pattern query. The choice of the optimal strategy is a distributed query optimization problem with interesting research questions.

5 Stream Calculus & Algebra

An essential aspect of our proposal is the seamless transition between explicit and intentional data, which is needed in order to support effectively the loose integration paradigm of the data ring. ActiveXML [6] was designed to capture such issues and this is why we choose it as the foundation for the topological layer. An ActiveXML document is an XML document where certain elements denote embedded calls to Web services. For instance, the company document may contain the CEO phone number as a Web service call.

A language in the style of ActiveXML should also serve as the basis for the physical layer. In particular, this language should permit the use of functions in both pull and push (subscription) modes, the processing of streams of data, and of recursion (because of the graph nature of the Web). As shown in a recent work [2], distributed query evaluation and optimization can be naturally captured using ActiveXML algebraic expressions, based on the exchange of distributed query execution plans. The expressions include standard algebraic XML operations and send/receive operators, all over XML streams. Note that these may be seen as particular workflow descriptions of a strong database flavor.

An important element in this context is the cooperation between local query evaluation (under the responsibility of local systems, perhaps relational systems) and global query evaluation that is the domain of the data ring. The data ring sees the local query processors (with their optimizers) as boxes with possibly different querying capabilities, in the same vein as mediation systems [14].

6 Autonomic Administration

As mentioned from the beginning, our vision is to enable non-experts, such as scientists, to create content sharing communities with minimal overhead. In turn, this clearly implies that the users should be alleviated from any administration duties, and hence the ring should function without the intervention of an administrator or any central authority. Moreover, the distributed nature of data rings suggests that this autonomic operation must be completely decentralized.

Our goal of autonomic administration clearly targets the physical layer of the data ring. Similar to previous proposals on autonomic computing, we envision in particular the following functionality: self monitoring (e.g., gathering automatically statistics), self tuning (e.g., selecting automatically indexes), or self healing (e.g., selecting automatically a substitute for some failing Web service). The notion of self-administration and most importantly self-tuning have already been explored in the context of relational databases. However, in the data ring context they become an absolute necessity because of the nature of the system, and they also acquire unique characteristics: autonomic administration is continuously "on" so that it can reconfigure the system in vu of volatile workloads; and, it requires the collaboration of the peers, whereas existing solutions on self-administration typically target a centralized scenario.

Clearly, the previous requirements set the bar very high. We believe that the power of parallelism that comes with a P2P system may compensate limitations of the technology. For instance, with many machines at our disposal, we can run in parallel more costly tuning algorithms that can meet some (or hopefully all) of the requirements that we have outlined.

7 Data Activation for Files

One of our basic assumptions is that a significant portion of the available data resides in file systems that do little effort to optimize their access or their updates. Still, we envision that the data ring should provide efficient declarative query capabilities over such sources, by means of *data self-activation*.

To illustrate this idea, let us consider a music catalog that has been published as a file in the ring and is queried heavily. Again, we assume that the catalog is presented as an XML view to the users, and queries are translated to a suitable file algebra. Since that physical organization cannot be changed, the data ring may decide, based on self-statistics, to reorganize the catalog in redundant physical structures that are more efficient to access. For instance, it may decide to replicate the catalog (or some part thereof) on a different peer. Another option is to replicate the catalog in a different physical system, e.g. an indexed relational database, in order to answer efficiently queries that access some specific attributes.

Clearly, data self-activation is closely linked to the previously described goals of autonomic administration and self-tuning in particular. For instance, the creation of redundant file indices can be seen as a form of physical design tuning. We choose to emphasize data self-activation as a separate point in the data ring design because scientific data management relies heavily on file systems, and we believe that special attention should be given to the idea of in-situ query processing for data stored in files.

8 Conclusion

The starting point of the data ring project is the observation that large communities of users will increasingly share content over the Internet. We propose the data ring as the means to bring the great benefits of database technology to these communities and to enable them to easily create, administer, and exploit shared content. We have outlined the general principles behind data rings, and discussed some research challenges met on the way to the realization of this vision.

The realization of data rings will clearly require the collaboration of numerous researchers, and we hope to entice other researchers to join us in this endeavor. Indeed, the ideas presented in this paper already lead to a joint project between several research groups, also called Dataring, and supported by the French Agence Nationale de Recherche.

References

[1] S. Abiteboul. Managing an XML Warehouse in a P2P Context. In *CAiSE*, pages 4–13, 2003.

[2] S. Abiteboul, I. Manolescu, and E. Taropa. A Framework for Distributed XML Data Management. In *10th International Conference on Extending Database Technology*, pages 1049–1058, 2006.

[3] S. Abiteboul, Ioana Manolescu, Neoklis Polyzotis, Nicoleta Preda, and Chong Sun. XML processing in DHT networks. In *Proceedings of IEEE ICDE 2008*, pages 606–615, 2008.

[4] S. Abiteboul and N. Polyzotis. The Data Ring: Community Content Sharing. In *Conference on Innovative Data Systems Research (CIDR)*, 2007.

[5] S. Agrawal, S. Chaudhuri, and V.Narasayya. Automated Selection of Materialized Views and Indexes for SQL Databases. In *Proceedings of the 26th Intl. Conf. on Very Large Data Bases*, pages 496–505, 2000.

[6] ActiveXML Web Site. `http://activexml.net`.

[7] Nicolas Bruno, Nick Koudas, and Divesh Srivastava. Holistic Twig Joins: Optimal XML Pattern Matching. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 310–321, 2002.

[8] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. (Available from `http://www.w3.org/TR/wsdl`), 2001.

[9] B. F. Cooper and H. Garcia-Molina. SIL: Modeling and Measuring Scalable Peer-to-Peer Search Networks. In *DBISP2P*, pages 2–16, 2003.

[10] B. F. Cooper and H. Garcia-Molina. Ad Hoc, Self-Supervising Peer-to-Peer Search Networks. *ACM Trans. Inf. Syst.*, 23(2):169–200, 2005.

[11] M. J. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4):27–33, 2005.

[12] L. Galanis, Y. Wang, S. R. Jeffery, and D. J. DeWitt. Locating Data Sources in Large Distributed Systems. In *Proceedings of the 29th Intl. Conf. on Very Large Data Bases*, 2003.

[13] P. Ganesan, Q. Sun, and H. Garcia-Molina. Adlib: A Self-Tuning Index for Dynamic Peer-to-Peer Systems. In *Proceedings of the 21st Intl. Conf. on Data Engineering*, pages 256–257. IEEE Computer Society, 2005.

[14] L.M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing Queries Across Diverse Data Sources. In *Proceedings of the 23rd Intl. Conf. on Very Large Data Bases*, pages 276–285, 1997.

[15] R. Huebsch, J.M. Hellerstein, N. Lanham, B.T. Loo, S. Shenker, and I. Stoica. Querying the internet with PIER. In *Proceedings of 19th International Conference on Very Large Databases*, 2003.

[16] V. Josifovski, P. Schwarz, L.M. Haas, and E. Lin. Garlic: a new flavor of federated query processing for DB2. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 524–532, 2002.

[17] P. Krishna and T. Johnson. Index replication in a distributed B-tree. In *COMAD*, 1994.

[18] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, 2001.

[19] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, 2001.

[20] I. Tatarinov, Z. G. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The piazza peer data management project. *SIGMOD Rec.*, 32(3):47–52, 2003.

The Social Future of Web Search: Modeling, Exploiting, and Searching Collaboratively Generated Content

Eugene Agichtein
Emory University
eugene@mathcs.emory.edu

Evgeniy Gabrilovich
Yahoo! Research
gabr@yahoo-inc.com

Hongyuan Zha
Georgia Institute of Technology
zha@cc.gatech.edu

Abstract

Social, or collaboratively generated content (CGC) is transforming how we seek and find information online: it is now a prominent part of the web information ecosystem, and a powerful platform for information seeking. The resulting archives of both the content and the context of the interactions contain valuable information that is often not available elsewhere, and can be helpful for the development of novel ranking algorithms, and natural language processing, text mining, and information retrieval techniques. We review machine learning techniques for modeling CGC, focusing on tasks such as learning to estimate content quality, relevance, and searcher intent and satisfaction with the retrieved results. We describe how this information can be incorporated into learning-based ranking methods for searching social media, and how CGC could be used to improve performance on key text mining and search tasks.

1 Introduction

Proliferation of the Internet and ubiquitous access to the Web enable millions of Web users to collaborate online on a variety of activities. Many of these activities result in the construction of large repositories of knowledge, either as their primary aim (e.g., Wikipedia) or as a by-product (e.g., Yahoo Answers). During the last few years, this user- or collaboratively-generated content (CGC) has started dominating the web: increasingly, users participate in content creation, rather than just consumption. Published and professional web content together is estimated to be generated at about 5 Gigabytes/day, whereas user-generated content is created at the rate of about 10 Gigabytes a day, and growing [RT08]. Popular user-generated content domains include blogs and web forums, folksonomies, social bookmarking sites, photo and video sharing communities, as well as social networking platforms such as Facebook and MySpace, which offer a combination of all of these with an emphasis on the relationships among the users of the community.

The unprecedented amounts of information in the collaboratively generated content sites, can enable new, knowledge-rich approaches to information access, which are significantly more powerful than the conventional word-based methods. Considerable progress has been made in this direction over the last few years. Examples include explicit manipulation of human-defined concepts and their use to augment the bag of words, using large-scale taxonomies of topics from Wikipedia or the Open Directory Project to construct additional class-based features, or using newly available word senses and examples of their usage for better disambiguation.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

At the same time, mining and organizing information in CGC is a challenging task. Unlike "traditional" web content, social media content is inherently dynamic: for hours or days after the initial posting of an item, users continue to post comments (e.g., on blogs and forums), vote for stories (e.g., on Digg), and contribute and rate answers to questions (e.g., on Yahoo! Answers). As responses and ratings arrive, the perceived popularity, quality, or even interpretation of an item may change significantly over time. Another important difference between collaboratively-generated and traditional content is the variance in the content quality. As Anderson [And06] describes, in traditional publishing—mediated by a publisher—the typical range of quality is substantially narrower than in niche, unmediated markets whereas social media content ranges from very high-quality items to low-quality, sometimes abusive content. This makes the tasks of filtering and ranking in CGC systems more complex. At the same time, social media presents inherent advantages over traditional collections of documents: the rich structure of CGC offers, in addition to document content and link structure, a wide variety of metadata about authorship, user feedback, as well as interactions with the content and other users.

Even the notion of "search" is not well understood in the social media context. For example, social media users often discover information by following recommendations from their social network (e.g., sites such as Facebook.com or Myspace.com), from other users on a site (e.g., sites such as Digg.com or Slashdot.org), by following some users explicitly (e.g., Twitter.com), or by contributing new content annotations or using existing tags contributed by other users (e.g., del.icio.us). These ways of finding and sharing information differ significantly from "traditional" keyword querying and relevance ranking models of web search. Thus, specialized techniques for modeling, analyzing, and searching in the social media environment have become an active area of research.

This paper surveys recent progress on modeling, exploiting, and searching collaboratively generated content. First, we review recent progress on modeling the process of social media creation, focusing on the user interactions, and the resulting quality of content. Then, we discuss how this content could be exploited to improve search-related tasks such as classification. Finally, we discuss recent work on incorporating this information to improve searching of social media and the web as a whole.

2 Modeling Collaboratively Generated Content

We first describe content creation in popular CGC sites, and some of the proposed models of content creation and evolution. Then, we review recent work on modeling content quality, both from the objective editorial perspective and that of user interest and engagement. These models are crucial for the work of exploiting and searching of social media content, described in later sections.

2.1 Content Creation

As representative examples, we consider Wikipedia, Youtube.com, and Yahoo! Answers web sites:

Wikipedia: Now a prototypical example of valuable collaboratively generated content, Wikipedia allows any user to create or edit virtually any page on the site, on any topic. The entire history of edits is preserved, allowing for reverting to an earlier version, and additional moderation is performed for controversial or popular topics. The central element of Wikipedia is an article, which serves as the unit of organization of the content, discussion comments, edits, and links. Numerous descriptive studies focused on Wikipedia content creation, including [CSC$^+$06, Gil05, WH07].

Collaborative Question Answering (CQA): Community-driven question/answering portals enable users to answer questions posed by other users, providing an alternative to automatic web search: rather than submitting keyword queries to search engines, users express detailed information needs, and often obtain direct responses from other users. In some markets, notably in South Korea, this information seeking behavior has eclipsed the use of traditional web search. In the United States, Yahoo! Answers has attracted approximately 100 million users, and has a growing archive of more than 400 million answers to questions, according to 2008 estimates. The resulting content is increasingly incorporated into main search results by all of the major search engines. For concreteness, we focus on the Yahoo! Answers CQA portal, but the observations and procedures are quite similar in other portals such as Naver and Baidu Knows. The central element of a CQA system are questions. Figure 1 shows a lifecycle of a question in CQA. It starts in an "open" state where it receives answers. Then at some point (decided by the asker, or by an automatic timeout in the system), the question is considered "closed", and can receive no further answers. At this stage, a "best" answer" is selected either by the asker or through a voting procedure by other users; once a best answer is chosen, the question is "resolved." The system is partially moderated by the community: any user may report another user's question or answer as violating the community guidelines (e.g., containing spam, adult-oriented content, copyrighted material, etc.). A user can also award a question a "star", marking it as an interesting question, and sometimes can vote for the best answer for a question, and can give to any answer a "thumbs up" or "thumbs down" rating, corresponding to a positive or negative vote, respectively. Recent studies describing the statistics of content generation in CQA include [ACD$^+$08, AZBA08].

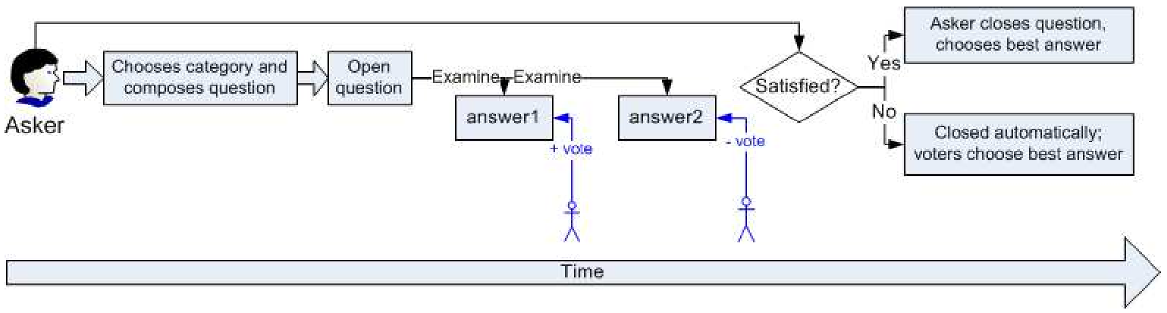

Asker
Chooses category and composes question
Open question
Examine
Examine
answer1
+ vote
answer2
- vote
Satisfied?
Yes
No
Asker closes question, chooses best answer
Closed automatically; voters choose best answer
Time

Figure 1: Question lifecycle in a typical CQA site

YouTube.com: As of 2009, Youtube.com is the largest user contributed video repository on the web, which allows comments, ratings, and annotations of the contributed videos by other users. Its central unit is a video clip, with many parallels to the CQA sites in that videos may be just a way to start a discussion amongst the users. Recent studies of Youtube.com content generation statistics include [CKR$^+$07, SH08].

The proposed models for content creation and attention can be roughly divided into visibility-based and preferential-attachment models. In visibility-based models, such as the model proposed by Lerman [Ler07], users are more likely to view, rate, and contribute to recently posted items, with participation exponentially decaying over time. Not surprisingly, the study shows a strong connection between a contributor's social network size, and content popularity. However, in some important sites such as Wikipedia or CQA, the social network is fluid and ad-hoc: users post content and interact independently of their social network connections. Leskovec et al. examined micro-evolution of a number of social media sites, and empirically evaluated variants of a preferential attachment model [LBKT08]. More recently, a large-scale study of both YouTube and Digg introduced and evaluated three families of content dynamics models [SH08]. Interestingly, one of the proposed models (namely, *Growth Profile*) describes the user contributions with a uniform accrual curve, which is appropriately re-scaled to account for the differences between item interestingness—without making any assumptions often built into exponential decay or other parametric models. In summary, modeling CGC creation is a young and active research area, with more accurate and insightful models in development.

2.2 CGC Accuracy and Quality

Because of the high variance of content quality in social media, estimation of content quality is an essential module for performing advanced information retrieval tasks. For instance, a quality score can be used as input to ranking algorithms or used to more effectively exploit CGC for web search tasks. Recent work by Agichtein et al. [ACD$^+$08] focused on the task of finding high quality content in social media. At the high level, their approach was to exploit features of social media that are intuitively correlated with quality, and then train a classier to select and weight the features for each specific type of item, task, and quality definition. Specifically, the authors modeled the intrinsic content quality, the interactions between content creators and other users, as well as the content usage statistics. All of these feature types were then used as an input to a classifier that can be tuned for the quality definition for the particular media type. The three classes of features included:

- **Intrinsic (text) content quality**: word n-grams, punctuation, capitalization, and spacing density, as well as the fraction of misspellings and typos. Also included were features capturing syntactic and semantic complexity, text grammaticality (estimated using part-of-speech sequences), and statistical properties of term distributions compared to reference corpora.

- **Social network features**: A significant amount of quality information can be inferred from the relationships between users and items. In particular, link analysis-based features were used, such as the hub and authority scores [K97], as well as the PageRank score of each item [BP98].

- **Viewing statistics**: Both first order statistics such as counts of how many times the content was viewed, as well as normalization by topic/genre popularity and temporal characteristics such as item posting time and "age" were used.

The resulting models, trained over quality judgments contributed by experts, performed on par with human editors—that is, the resulting accuracy was comparable to the inter-editor agreement between experts. They investigated the contributions of the different sources of quality evidence, have shown that some of the sources are complementary (i.e., capture the same high-quality content using the different perspectives). The combination of several types of sources of information is likely to increase the classifier's robustness to spam, as an adversary would required to not only create content the deceives the classifier, but also simulate realistic user relationships or usage statistics.

2.3 CGC Popularity and Usefulness

Another important characteristic of collaboratively generated content is popularity. Recent work on predicting content popularity on sites such as Youtube.com [SH08] or Digg.com [Ler07] exploited both the temporal dynamics of the interactions, as well as the social network structure, demonstrating that it is possible to identify items that will eventually become popular, as early as 30 minutes or an hour after the initial posting.

In the context of CQA, a more appropriate metric is *user satisfaction*: whether an asker in CQA is *satisfied* with the answers contributed by the community [LBA08]. This is in contrast to the more traditional relevance-based assessments that are often done by judges different from the original information seeker, which may result in ratings that do not agree with the target user. Nowhere does the problem of subjective relevance arise more prominently than within CQA, where many of the questions are inherently subjective, complex, ill-formed, or often all of the above. Not surprisingly, user's previous interactions such as questions asked and ratings submitted are a significant factor for predicting satisfaction. We hypothesized that asker's satisfaction with contributed answers is largely determined by the asker expectations, prior knowledge and previous experience with using the CQA site.

The features used for prediction are similar to the editorial quality above, including the question text: words and 2-word phrases in the question, the wh-type (e.g., what or where), and the length of the subject (title) and detail (description) of the question; Question-Answer Relationship, including overlap between question and answer text, answer length, and the number of candidate answers; Community-ratings, including "thumbs up", "thumbs down", the asker's previous contribution history; the answerer reputation; and the statistical word distributions of the question and answer text.

A variety of classification algorithms were applied to learn to predict asker satisfaction with the obtained responses. Interestingly, the automatic system was significantly more accurate than human raters on this task. One possible explanation is that human raters were not able to take advantage of the context in which the question was asked, or the asker's expectations and prior experience with the site. The implications of these results are that in order by carefully modeling the CGC creation process, automatic techniques can be build to evaluate and filter this content with accuracy as high as, or higher, then human experts.

Having described how CGC is created, and the models of the resulting content quality and popularity, we now consider how this data could be used to improve search and information access tasks.

3 Exploiting CGC for Information Access

Since collaboratively generated content is contributed by humans, it naturally embodies large amounts of human knowledge about the world. Back in the early years of AI research, Buchanan & Feigenbaum [BF82] formulated the *knowledge as power hypothesis*, which postulated that "The power of an intelligent program to perform its task well depends primarily on the quantity and quality of knowledge it has about that task." When computer programs face tasks that require human-level intelligence, such as intelligent information retrieval, it is only natural to use repositories of human knowledge to endow the machines with the breadth of knowledge available to humans. In this section, we survey a number of techniques we recently developed for using collaboratively generated content to improve information access.

3.1 Explicit Semantic Analysis

Observe that repositories of collaboratively generated content usually contain textual information rather than highly structured knowledge such as, for example, CYC [LG90]. There are several ways to extract knowledge from CGC. One way to use such content is simply to view it as additional training data, which can be used in unsupervised or semi-supervised scenarios [AZ05, NMTM00, Joa99]. Alternatively, a transfer learning approach can be employed to learn from the labels (e.g., tags or categories) associated with the collaboratively generated content and then reuse the learned models across different but related learning tasks [BDH03, DN05]. Yet another, and perhaps most promising direction, is to use the knowledge encoded in CGC repositories in order to construct new features that enrich the language of text representation.

Feature generation techniques were found useful in a variety of machine learning tasks [MR02, Faw93, Mat91]. These techniques search for new features that describe the target concept better than the ones supplied with the training instances. The key observation is that many cases textual objects in CGC repositories are (explicitly or implicitly) associated with knowledge concepts. Examples of such concepts include nodes of the Open Directory (`dmoz.org`), titles of Wikipedia (`wikipedia.org`) articles, or tags in FLICKR (`flickr.com`) or DEL.ICIO.US (`del.icio.us`). To this end, we proposed to learn a *semantic interpreter* that identifies these concepts in input texts and quantifies the degree of affinity of each concept to the input text. Subsequently, new features corresponding to these concepts enrich (or even replace) the conventional bag-of-words representation of the input text. We call our method Explicit Semantic Analysis (ESA), as it uses knowledge concepts explicitly defined and manipulated by humans.

Let us illustrate the value of such knowledge with a couple of examples. Without using external knowledge (specifically, knowledge about financial markets), one can infer little information from a very brief news title "Bernanke takes charge". However, using the algorithm we developed for consulting Wikipedia, we find that the following concepts are highly relevant to the input: BEN BERNANKE, FEDERAL RESERVE, CHAIRMAN OF THE FEDERAL RESERVE, ALAN GREENSPAN (Bernanke's predecessor), MONETARISM (an economic theory of money supply and central banking), INFLATION and DEFLATION. As another example, consider the title "Apple patents a Tablet Mac". Without deep knowledge of hi-tech industry and gadgets, one finds it hard to predict the contents of the news item. Using Wikipedia, we identify the following related concepts: APPLE COMPUTER, MAC OS (the Macintosh operating system) LAPTOP (the general name for portable computers, of which Tablet Mac is a specific example), AQUA (the GUI of MAC OS X), IPOD (another prominent product by Apple), and APPLE NEWTON (the name of Apple's early personal digital assistant). In both cases, the concepts identified by ESA provide valuable relevant knowledge that can greatly help in interpreting and processing the original input text.

For ease of presentation, in the above examples we only showed a few Wikipedia concepts identified by ESA as the most relevant for the input. However, the essence is representing the meaning of text as a weighted combination of multiple knowledge concepts (e.g., *all* Wikipedia concepts). Then, depending on the nature of the task at hand we either use these entire vectors of concepts, or use a few most relevant concepts to enrich the bag of words representation.

The ESA method imposes several simple requirements on a suitable knowledge repository:

1. It should be comprehensive enough to include concepts in a large variety of topics.
2. It should be constantly maintained so that new concepts can be promptly added as needed.
3. Since the ultimate goal is to interpret *natural* language, we would like the concepts to be *natural*, that is, concepts recognized and used by human beings.
4. Each concept should have associated text, which would serve as training example(s) for learning to recognize this concept in input texts.

Creating and maintaining such a set of natural concepts requires an enormous effort by many people, but fortunately this is exactly how CGC repositories are constructed and maintained. In an empirical evaluation, ESA-based semantic interpreters were implemented using two specific CGC repositories, namely, the Open Directory Project and Wikipedia. Using knowledge-rich features generated based on these repositories led to significant improvements in text categorization [GM07, GM09] and information retrieval [EGM08].

We believe the most important aspects of ESA are its ability to address synonymy and polysemy, which are arguably the two most important problems in natural language processing. Thus, two texts can discuss the same topic using different words, and the conventional bag of words approach will not be able to identify this commonality. On the other hand, the mere fact that the two texts contain the same word does not necessarily imply that they discuss the same topic, since that word could be used in the two texts in two different meanings. Our concept-based representation allows generalizations and refinements of word meaning, and partially address synonymy and polysemy.

The empirical evaluation showed that particularly large improvements in classifying short texts and in answering short queries. This behavior was expected, since external knowledge is required for interpreting short input texts correctly. The ESA technique also helped achieve state of the art results in computing semantic relatedness of natural language texts. In the extreme case—when computing relatedness of individual words—the bag of words approach is simply not applicable (except for the trivial case when the two words to be compared are identical). Consequently, to compute semantic relatedness of texts we completely replace their bag-of-words representation with the concept-based one. Notably, using a larger fraction of the available concepts led to a more fine-grained representation and consequently to a better estimation of semantic relatedness (as measured by higher correlation with gold-standard human judgments).

3.2 Domain-specific query augmentation using folksonomy tags

Folksonomy is a method for assigning user-defined labels to objects stored in public repositories of textual or multimedia content. Examples of popular folksonomies include FLICKR (a photo collection), DEL.ICIO.US (a bookmark sharing project), and YOUTUBE (a video sharing system). Typically, users can add tags to any object, whether they "own" it or not. Folksonomies facilitate interaction between Web users and promote knowledge sharing by integrating the user-defined tags in searching and browsing activities. In a sense, folksonomies comprise an alternative to restricted lexicons, as the numerous tags potentially allow users to achieve higher recall. When the original content creator might not have thought of all the applicable tags, users who subsequently encounter the object are likely to add tags they deem relevant.

In a recent study [BCG$^+$09], co-tagging (i.e., tagging of the same object with different labels) was used to infer tag relatedness *in the context* of individual folksonomies. The key contribution was an alternative definition of context as a collection of tags assigned to or related to an object. Given a query, the system first identifies a set of relevant tags, and then uses tags that co-occur with them to augment the query. This method performs domain-specific query disambiguation, and can actually learn that a query "menu" is likely to have food connotation on FLICKR but user interface connotation on DEL.ICIO.US.

This method was implemented for context-sensitive query augmentation using contextual advertising as an application domain. Note that a query submitted to FLICKR most likely conveys a different intent than the same query submitted to DEL.ICIO.US. That is, knowing at which site the query is submitted can help identify the user's search intent. Treating the content of the site as the context for queries, and matching ads accordingly, can potentially improve user experience. In the previous example, FLICKR ads for the query "menu" should ideally include offers from restaurants rather than services of UI experts, which would be more appropriate on DEL.ICIO.US. Having computed site-specific tag co-occurrence statistics, the relevant tags are used to expand the bag of words for the query, as well as classify those tags to create new taxonomy-based features. Thus, using CGC to represent queries in this rich feature space results in more relevant ad matches, so that the ads displayed on different folksonomy sites better reflect the intent of their users.

4 Searching Collaboratively Generated Content and the Web

To make CGC more accessible and serving a broad spectrum of users, it is important to provide an effective search interface. As discussed, CGC is different from the traditional content on the web in style, quality, and authorship. More importantly, the explicit support for social interactions between users, such as posting comments, rating content, and responding to questions and comments makes social media unique and requires new techniques for searching. In this section, we use searching collaborative question answering (CQA) archives as a concrete example to discuss the various issues involved in searching CGC, and the prominent problem of how to combine CGC and the generic Web in a unified search process.

4.1 Learning to Rank CGC

Question-Answering (henceforth QA) is a form of information retrieval where the users' information need is specified in the form of a natural language question, and the desired result is self-contained *answer* (not a list of documents). QA has been particularly amenable to social media, as it allows a potentially more effective alternative to web search by directly connecting users with the information needs to users willing to share the information. In CQA portals such as Yahoo! Answers and Naver, users can express specific information needs by posting questions, and get direct responses authored by other users.

At CQA sites both the questions and responses are stored for future use by allowing searchers to first attempt to locate an answer to their question, if the same or similar question has been answered in the past. As QA portals grow in size and popularity, searching for existing answers for a given question becomes increasingly crucial to avoid duplication, and save time and effort for the users. Finding relevant questions and answers of a new query in QA archives, however, is a difficult task that is distinct from web search and web question answering, exemplified by the availability of explicit user feedback and interactions, explicit authorship and attribution information, and organization of the content around topical categories and past question threads. Because of the increasing popularity of Yahoo! Answers, Naver and other CQA sites, several recent research efforts tried to address the CQA search problems. Jeon et al. [JCL05, JCLP06] presented retrieval methods based on machine translation models to find similar questions from Naver. An alternative approach is possible, based on the general framework of learning to rank, seamlessly integrating the interaction features into the question and answer retrieval [BLAZ08b, BLAZ08a]. The learning to rank approach used was based on gradient boosting [Fri01, ZZCS07]. In the following we focus on the specific issues in extracting features that incorporate user interactions.

The approach abstracts the social content in CQA sites as a set of QA pairs. Given a user query, the goal is to order the set of QA pairs according to their relevance to the query, and the ordering is done by learning a ranking function for triples $(query, question, answer)$. Users in CQA sites not only ask and answer questions, but also actively participate in regulating the system. A user can vote for answers of other users, mark interesting questions and even report abusive behavior. Therefore, a CQA user has a threefold role: asker, answerer and evaluator. And there are respectively three types of user interaction activities: asking, answering and evaluating. Following the general practice in information retrieval, we can represent each query-question-answer triple $(query, question, answer)$ as a combination of textual features (i.e., textual similarity between query, question and answers), statistical features (i.e., independent features for query, question and answers) and social features (i.e., user interaction activities and community-based elements). In CQA sites, there is an additional important type of user feedback — user evaluation in the form of votes (represented as the "thumbs up" and "thumbs down" metaphors). We can use this information to infer preference relevance judgments for the set of answers.

We now elaborate on the social features: For each user in the user community of a CQA site, there are several features to describe his or her activities, such as the number of questions he or she asked, the number of answers he or she posted, the number of best answers he or she posted etc. These features to certain extent can approximate the user's expertise in the CQA site. And user's expertise within certain topics can in turn indicate the quality of his or her answers to the questions about certain topics. For example, in a query-question-answer triple, if answerer tend to post useful answers or even best answers in the past, he or she is more likely to give answers of high quality this time. Similarly reputation of askers and evaluators can also indicate quality of answers. Therefore, in each query-question-answer triple, we also extract features indicating user's activities in the CQA site such as "number of questions the asker asked in the community", "number of best answers the answerer posted in community", etc.

Empirical evaluations using Yahoo! Answers showed that textual features and community features are crucial, and that user feedback, while noisy, provides sufficient relevance information to improve the learning of the ranking functions. Interestingly, textual features were found less important for Precision at 1. The learning to rank method coupled with user interaction features achieves a significant improvement on the performance of QA retrieval over the Yahoo! Answers' default ranking and the supported optional votes-based ranking.

We emphasized that CQA sites heavily rely on active user participation, such as voting or rating of responses to a question. This user feedback is invaluable for ranking, filtering, and retrieving high quality content as we discussed above. Unfortunately, as collaborative question answering, and CGC in general, move into the mainstream and gain in popularity, the quality of the user feedback degrades. Some of this is due to noise, but, increasingly, a small fraction of malicious users are trying to "game the system" by selectively promoting or demoting content for profit, or fun. Hence, an effective ranking of CGC content must be robust to noise in the user interactions, and in particular to vote spam. Bian et al. [BLAZ08a] extend the learning to rank idea and consider general vote spam attack models. In particular, a new method of training a ranker was introduced, in order to increase its robustness to common vote spam attacks. The results of a large-scale experimental evaluation show that such a ranker is significantly more robust to vote spam, compared to a state-of-the-art baseline, as well as the ranker not explicitly trained to handle malicious interactions.

4.2 Unified ranking of CGC and Web documents

CGC and the Web are often complementary. It would benefit the users if a single search interface can be developed which provide a unified ranking of the contents of both. To this end, we consider a learning to rank framework that merges general Web search results with CGC results. One key observation is that the two ranking problems have very different data distributions and they also use different feature sets. For example, user interaction activities and community-based elements such as user votes are not available for Web search results. Therefore we can't rely on commonality between the learning tasks other than 1) they both rank documents by assigning scores to documents, and 2) that their *relevance labels* are comparable.

We advocate the approach to *simultaneously* learn two ranking functions, $h_W(\cdot)$ for Web documents and $h_A(\cdot)$ for CGC. We add preference constraints between documents across different rankings. These constraints usually involve only documents related to the same queries. With our assumption on consistent labels, they can be induced by these labels. For example, a *Good* document in Web ranking list one should be ranked higher than a *Fair* document in CGC ranking list. Suppose x and y are feature vectors of two documents related to the same query but are in the ranking lists of Web search and CQA, respectively. When training h_W and h_A, we add constraint $h_W(x) \geq h_A(y)$, if we want to rank x higher than y in the merged lists. This requires that the learning algorithms for h_W and h_A be able to take this kind of constraints into account, or soft versions of such constraints using a gradient boosting-based ranking algorithm [Fri01, ZZCS07]. In addition to merging result lists, significant work is required to group and display the generic and social media search results appropriately and intuitively, which remains an active research area as new user models for search in social media emerge.

5 Conclusions

We believe that collaboratively generated content (CGC) sites, both as the platforms for information seeking and sharing, and as the source of human knowledge, have the potential to revolutionize information access. Accomplishing this requires new methods for modeling and processing this content, as well as techniques for incorporating information from CGC into search. In this article we presented a brief overview of recent progress on these exciting research directions.

ACKNOWLEDGMENTS: The first author acknowledges generous support from the Yahoo! Faculty grant, the Cisco "Improving the Human Network" program, and the Emory College Seed Fund.

References

[ACD$^+$08] E. Agichtein, C. Castillo, D. Donato, A. Gionis, and G. Mishne. Finding high-quality content in social media with an application to community-based question answering. In *Proceedings of WSDM*, 2008.

[And06] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[AZ05] Rie Kubota Ando and Tong Zhang. Framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, pages 1817–1853, 2005.

[AZBA08] Lada A. Adamic, Jun Zhang, Eytan Bakshy, and Mark S. Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 665–674, 2008.

[BP98] Sergey Brin and Larry Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th International World Wide Web Conference*, pages 107–117, 1998.

[BCG$^+$09] Andrei Broder, Peter Ciccolo, Evgeniy Gabrilovich, Vanja Josifovski, Donald Metzler, Lance Riedel, and Jeffrey Yuan. Online expansion of rare queries for sponsored search. In *Proceedings of the 18th International World Wide Web Conference*, 2009.

[BDH03] Paul N. Bennett, Susan T. Dumais, and Eric Horvitz. Inductive transfer for text classification using generalized reliability indicators. In *Proceedings of the ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*, 2003.

[BF82] B. G. Buchanan and Edward Feigenbaum. Forward. In R. Davis and Douglas Lenat, editors, *Knowledge-Based Systems in Artificial Intelligence*. McGraw-Hill, 1982.

[BLAZ08a] J. Bian, Y. Liu, E. Agichtein, and H. Zha. A few bad votes too many? towards robust ranking in social media. In *The 4th International Workshop on Adversarial Information Retrieval on the Web*, 2008.

[BLAZ08b] J. Bian, Y. Liu, E. Agichtein, and H. Zha. Finding the right facts in the crowd: Factoid question answering over social media. In *Proc. of 17th International World Wide Web Conference (WWW2008)*, 2008.

[CKR$^+$07] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 1–14, 2007.

[CSC$^+$06] A. Capocci, V. D. P. Servedio, F. Colaiori, L. S. Buriol, D. Donato, S. Leonardi, and G. Caldarelli. Preferential attachment in the growth of social networks: the case of wikipedia, 2006.

[DN05] Chuong Do and Andrew Ng. Transfer learning for text classification. In *Proceedings of Neural Information Processing Systems (NIPS)*, 2005.

[EGM08] Ofer Egozi, Evgeniy Gabrilovich, and Shaul Markovitch. Concept-based feature generation and selection for information retrieval. In *AAAI'08*, July 2008.

[Faw93] Tom Fawcett. *Feature Discovery for Problem Solving Systems*. PhD thesis, UMass, May 1993.

[Fri01] J. Friedman. Greedy function approximation: a gradient boosting machine. In *Ann. Statist.*, 2001.

[Gil05] Jim Giles. Internet encyclopaedias go head to head. *Nature*, 438:900–901, 2005.

[GM07] Evgeniy Gabrilovich and Shaul Markovitch. Harnessing the expertise of 70,000 human editors: Knowledge-based feature generation for text categorization. *Journal of Machine Learning Research*, 8:2297–2345, October 2007.

[GM09] Evgeniy Gabrilovich and Shaul Markovitch. Wikipedia-based semantic interpretation for natural language processing. *Journal of Artificial Intelligence Research*, pages 443–498, 2009.

[JCL05] J. Jeon, W.B. Croft, and J.H. Lee. Finding similar questions in large question and answer archives. In *Proceedings of CIKM*, 2005.

[JCLP06] J. Jeon, W.B. Croft, J.H. Lee, and S. Park. A framework to predict the quality of answers with non-textual features. In *Proceedings of SIGIR*, 2006.

[Joa99] Thorsten Joachims. Transductive inference for text classification using support vector machines. In *Proceedings of the 13th International Conference on Machine Learning*, 1999.

[K97] Jon Kleinberg. Authoritative Sources in a Hyperlinked Environment. *Journal of the ACM*, 46(5):604–632, 1997.

[LBA08] Yandong Liu, Jiang Bian, and Eugene Agichtein. Predicting information seeker satisfaction in community question answering. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2008)*, 2008.

[LBKT08] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470, 2008.

[Ler07] Kristina Lerman. Social information processing in social news aggregation. *IEEE Internet Computing: special issue on Social Search*, 2007.

[LG90] Douglas Lenat and R. Guha. *Building Large Knowledge Based Systems*. Addison Wesley, 1990.

[Mat91] Christopher J. Matheus. The need for constructive induction. In L.A. Birnbaum and G.C. Collins, editors, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 173–177, 1991.

[MR02] Shaul Markovitch and Danny Rosenstein. Feature generation using general constructor functions. *Machine Learning*, 49(1):59–98, 2002.

[NMTM00] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2-3):103–134, 2000.

[RT08] Raghu Ramakrishnan and Andrew Tomkins. Toward a PeopleWeb. *Computer*, 40(8):63–72, August 2007.

[SH08] Gábor Szabó and Bernardo A. Huberman. Predicting the popularity of online content. *CoRR*, 2008.

[WH07] Dennis M. Wilkinson and Bernardo A. Huberman. Cooperation and quality in wikipedia. In *WikiSym '07: Proceedings of the 2007 international symposium on Wikis*, pages 157–164, 2007.

[ZZCS07] Z. Zheng, H. Zha, K. Chen, and G. Sun. A regression framework for learning ranking functions using relative relevance judgments. In *Proc. of SIGIR*, 2007.

Very Large Data Bases

VLDB 2009

CALL FOR PARTICIPATION

35th International Conference on
Very Large Data Bases
24-28th August 2009,
Lyon, France

http://vldb2009.org/

VLDB is a premier international forum for database researchers, vendors, practitioners, application developers, and users. We are looking forward to an exciting conference, discussing original results on all aspects of data management, tutorials, demonstrations, and for the first time a track on experiments and analyses that will present the most critical issues and views on practical leading-edge database technology, applications, and techniques. VLDB 2009 also hosts a series of workshops that cover important aspects in the thematic context of databases.

Schedule:

Monday, 24th August: **Workshops**
Tuesday to Thursday, 25 - 27th August: **Main Conference**
Friday, 28th August: **Workshops**

Please note the days of week of the Main Conference!

Early registration deadline: 15th of July 2009
Information on the conference and registration is available at: **http://vldb2009.org**

See you in Lyon!

VLDB 2009 organization committee

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398