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

Copyright 2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

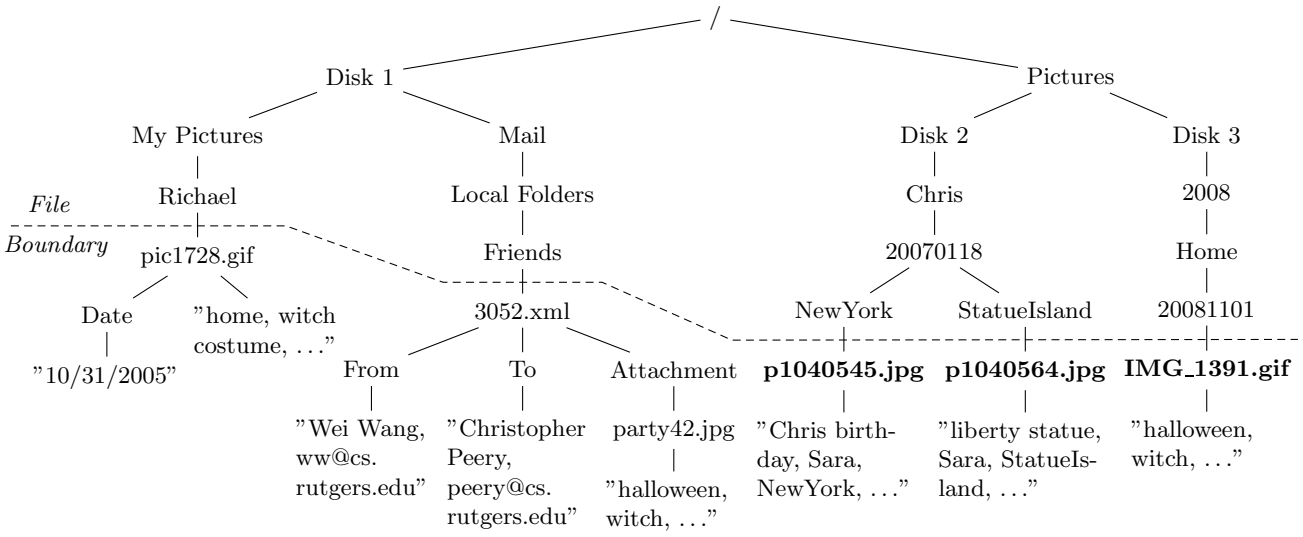

Figure 1: A subset of a user personal information file structure.

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:

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

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

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.

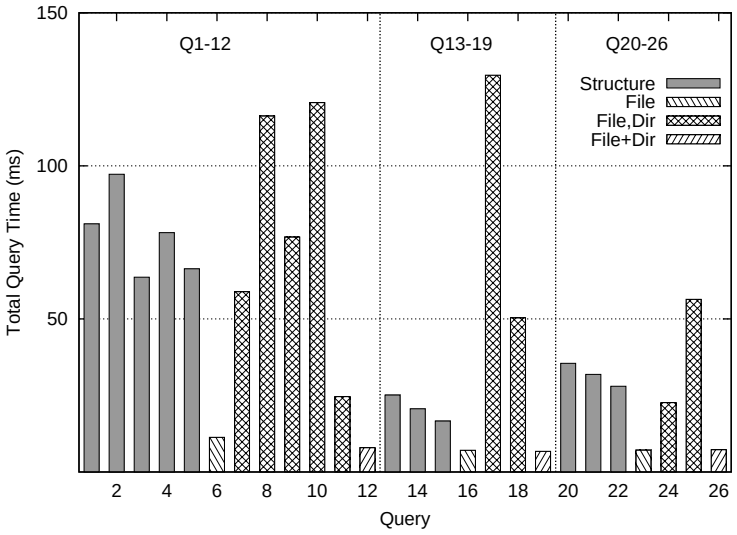

Figure 2: Query Processing time.

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/`.