11-12-2009

Efficient Storage and Domain-Specific Information Discovery on Semistructured Documents

Fernando R. Farfan
Florida International University, ffarfan@cis.fiu.edu

DOI: 10.25148/etd.FI09120826

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

EFFICIENT STORAGE AND DOMAIN-SPECIFIC INFORMATION DISCOVERY ON SEMISTRUCTURED DOCUMENTS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Fernando Farfán

2009

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Fernando Farfán, and entitled Efficient Storage and Domain-Specific Information Discovery on Semistructured Documents, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Peter Clarke

Raju Rangaswami

Naphtalie Rishe

Debra VanderMeer

Evangelos Christidis, Major Professor

Date of Defense: November 12, 2009

The dissertation of Fernando Farfán is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean George Walker
University Graduate School

Florida International University, 2009

DEDICATION

To my loving wife, Claudia.

To my children, Javier and Mariana.

To my parents, for all their support during all this time.

Ad Maiorem Dei Gloriam.

ACKNOWLEDGMENTS

I would have never been able to finish my dissertation without the guidance of my advisor, committee members, help from friends, and support from my wife, children and family.

I would like to express my deepest gratitude to my advisor Dr. Vagelis Hristidis, for his excellent guidance, example, patience and support. Also, to Dr. Raju Rangaswami, who along with Dr. Hristidis gave me the opportunity to join his research projects that led me beyond a Master's program and finalize with this dissertation. I would also like to thank Dr. Debra VanderMeer, Dr. Peter Clarke and Dr. Naphtalie Rishe, for guiding my research and being part of my dissertation committee.

I want to thank the Institute of International Education and the Fulbright Program, for granting me the honor of being a Fulbright Scholar during my Masters program at FIU. Also, I would like to thank the School of Computing and Information Sciences at FIU, for providing me with the appropriate funding opportunities to pursue my studies and research, to the University Graduate School for granting me a Dissertation Year Fellowship for my last year of work.

To all my friends in the Databases and Systems Research Laboratory at FIU: Medha, Ramakrishna, Luis, Ricardo, Jorge, Sajib, Eduardo and Theodore. Thank you for your friendship, support and teachings.

I would also like to thank my parents, grandparents, brother and sister, for being always there supporting me and encouraging me to keep going.

Finally, I would like to thank my wife Claudia, and our children Javier and Mariana, for being always my strength and inspiration, for cheering me up, and patiently standing by me throughout these times.

ABSTRACT OF THE DISSERTATION

EFFICIENT STORAGE AND DOMAIN-SPECIFIC INFORMATION DISCOVERY ON SEMISTRUCTURED DOCUMENTS

by

Fernando Farfán

Florida International University, 2009

Miami, Florida

Professor Evangelos Christidis, Major Professor

The increasing amount of available semistructured data demands efficient mechanisms to store, process, and search an enormous corpus of data to encourage its global adoption. Current techniques to store semistructured documents either map them to relational databases, or use a combination of flat files and indexes. These two approaches result in a mismatch between the tree-structure of semistructured data and the access characteristics of the underlying storage devices. Furthermore, the inefficiency of XML parsing methods has slowed down the large-scale adoption of XML into actual system implementations. The recent development of lazy parsing techniques is a major step towards improving this situation, but lazy parsers still have significant drawbacks that undermine the massive adoption of XML.

Once the processing (storage and parsing) issues for semistructured data have been addressed, another key challenge to leverage semistructured data is to perform effective information discovery on such data. Previous works have addressed this problem in a generic (i.e. domain independent) way, but this process can be improved if knowledge about the specific domain is taken into consideration.

This dissertation had two general goals: The first goal was to devise novel techniques to efficiently store and process semistructured documents. This goal had two specific aims: We proposed a method for storing semistructured documents that maps the physical characteristics of the documents to the geometrical layout of hard drives. We developed a Double-Lazy Parser for semistructured documents which introduces lazy behavior in both the pre-parsing and progressive parsing phases of the standard Document Object Model's parsing mechanism.

The second goal was to construct a user-friendly and efficient engine for performing Information Discovery over domain-specific semistructured documents. This goal also had two aims: We presented a framework that exploits the domain-specific knowledge to improve the quality of the information discovery process by incorporating domain ontologies. We also proposed meaningful evaluation metrics to compare the results of search systems over semistructured documents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

CHAPTER 1

INTRODUCTION

Semistructured data models have become popular and widely adopted as an effective means to encode and exchange documents in heterogeneous environments. An increasing number of applications manage large amounts of semistructured data [PGMW95]. Such applications include Bioinformatics suffix-tree-based sequence alignments [DKF$^+$99], genomics data analysis [Rok07], multi-resolution video [FJS96], clinical data [DAB$^+$06], XML Databases, and even directory-file hierarchies in general-purpose systems. Moreover, hundreds of application languages have based their specification in semistructured formats. Some examples include Medical Markup Language (MML) [MML08], Geographic Information Systems Markup Language (GML) [GML08], Open Document Format (ODF) [ope08, oox08], Health Level 7 [HL708a], and Scalable Vector Graphics (SVG) [SVG08].

To support and further encompass this wide adoption of semistructured data formats, specifically represented by the eXtensible Markup Language XML [BPSM$^+$06], efficient mechanisms to store, parse, and search such documents are necessary. These three phases constitute a line of work that needs to be optimized to ensure the highest performance and quality in processing semistructured documents. Figure 1.1 shows these three phases. A large amount of research has been driven to improve these three processing phases. But there is still room for improvement.

Current approaches to store semistructured data either map the data to an underlying relational database system (e.g., [BFRS02a, DFS99, MH04, RFHR, STZ$^+$]), or use the abstraction provided by a general-purpose object storage manager [CDF$^+$94], or use a combination of flat files and indexes (e.g., [AGM$^+$90, Gal07, JAKC$^+$02, KM06, Xal07, XT07]). These storage schemes, however, ignore the mismatch between the structure and navigational primitives of semistructured data and the access characteristics of disk

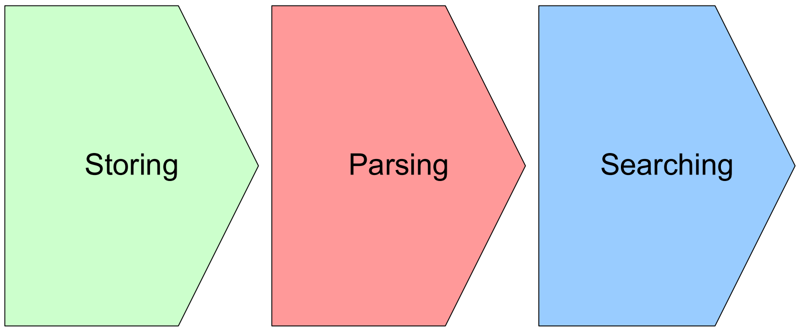


Figure 1.1: Three phases to work with semistructured documents.

drives. In particular, semistructured data have a tree (or graph) structure with tree-type operations. Relational databases, on the other hand, store structured tables that are optimized for row-based access, and flat files are unstructured, optimized for sequential access. Further complicating this mismatch, the underlying storage device, i.e. disk drives, store information in circular tracks that are accessed with mechanical seek and rotational overhead. These current solutions result in sub-optimal accesses to semistructured data. Given the abundance of semistructured data today, there is an immediate need for re-examining the current storage and access machinery. In this thesis, I explore strategies to optimize the storage, processing, and retrieval of semistructured data on disk drives by explicitly accounting for the mismatch between the structure of the data and the disk drive characteristics.

A key step in the massive adoption of semistructured data is the optimization of its processing mechanisms. The importance of efficient XML parsing methods has been underscored by Nicola and John [NJ03]; they showed that the parsing process when using the Document Object Model (DOM) [DOM08] is processor and memory consuming, particularly needing main memory as much as five times the size of the original document. Lazy XML parsing has been proposed (e.g., [xer08]) to improve the performance of the parsing process by avoiding the loading of unnecessary elements. This is a significant improvement. However, it still requires initial preprocessing phases during which the whole document has to be processed. It is necessary to develop new techniques that exploit the physical layout of semistructured documents in order to further optimize the parsing process on semistructured documents.

Although a vast corpus of work [CKKS05, FG01, CMKS03, CMM$^+$03, GSBS03, HPB03, HP06, LYJ04, XP05] has addressed the problem of quality Information Retrieval (IR) on semistructured data, a series of challenges arise when the search process is performed over domain-specific documents. The definition and structure of queries, search algorithms, and results should embrace and resemble as much knowledge about the specific nature of the documents as possible.

This thesis presents new techniques to improve the performance and quality of the three phases presented in Figure 1.1, summarized in the following aims:

i Exploit the physical organization and layout of semistructured documents to obtain a more efficient storage mechanism,

ii Efficiently parse and process semistructured documents by skipping unnecessary data,

iii Perform domain-specific Information Discovery by studying the semantics of the structure and the content of the documents for various domains, and

iv Design meaningful evaluation metrics for search systems that deal explicitly with collections of semistructured documents.

The rest of this thesis is organized as follows. The next chapter presents the research significance of this dissertation. Chapter 3 presents the data models and background considerations used in the rest of the chapters. The first part of the thesis discusses the efficient storage of semistructured data and is found in Chapter 4. The second part treats the efficient parsing of semistructured documents and is found in Chapter 5. The third part discusses challenges and techniques to provide domain-specific information discovery on semistructured documents and is developed in Chapters 6, 7 and 8. The conclusions to the dissertation are found in Chapter 9.

In each chapter we motivate the need for the work and present relevant background material. We then present the theoretical model and our research approach to solve the specific problems. We then show the experimental analysis of the introduced techniques. The related work for the research is presented next, followed by the chapter's conclusions.

CHAPTER 2

RESEARCH SIGNIFICANCE

The significance of this research is as follows:

1. Semistructured documents have been adopted in many environments. From large database installations to myriads of languages and dialects based on the Extensible Markup Language (XML), semistructured documents can be found everywhere. However, the current technologies to store, process and search this type of data have not reached an optimal level of performance.

2. Current approaches to store semistructured data either map the data to an underlying relational database system, use the abstraction provided by a general-purpose object storage manager, or use a combination of flat files and indices. Since these approaches retrofit existing storage mechanisms to work with semistructured data, their scope is restricted to the underlying mechanisms which are predominantly optimized for sequential accesses resulting in a mismatch between the structure and navigational primitives of semistructured data and the access characteristics of disk drives. Given the growing amount of semistructured data, there is a need for re-examining the current storage and access machinery that support them, and to design strategies to optimize the storage and retrieval of semistructured data on disk drives by explicitly accounting for the mismatch between the structure of the data and the disk drive storage and access characteristics.

3. The widespread use of semistructured documents, and in particular XML, requires efficient parsing techniques. The importance of efficient methods for parsing XML documents was underscored by Nicola and John [NJ03]; they showed that the parsing process is processor and memory consuming, particularly needing main memory as much as five times the size of the original document. This prohibitive re-

quirement makes imperative to develop more efficient mechanisms for parsing and processing.

4. As semistructured documents become more popular and widespread, so does the need for efficient and high-quality tools for searching and discovering information over these document corpora. Although several efforts have been made to optimize search systems for semistructured documents and XML repositories, it is possible to improve the quality of these systems by integrating into the search process the knowledge of the particular domain. While previous solutions exploit the structural and syntactical features of XML, we need to exploit semantic features, user preferences, and other domain knowledge that is captured and referenced by the documents.

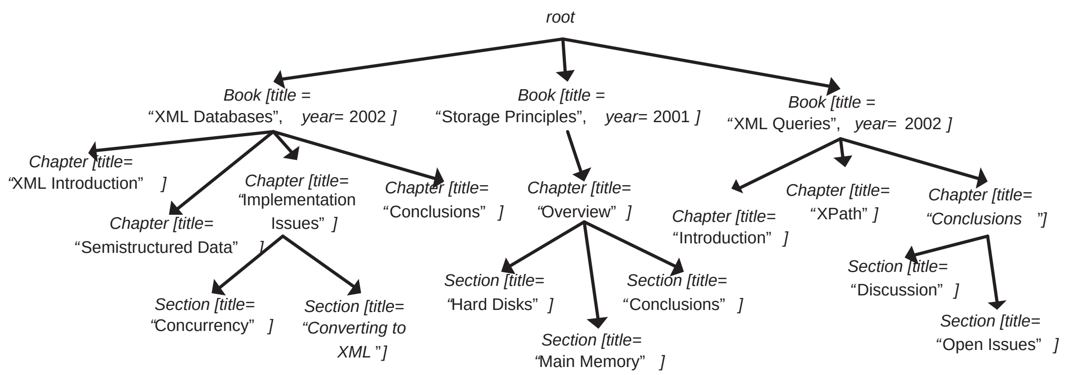
CHAPTER 3

BACKGROUND

In this chapter we establish some notation and initial definitions that will be of interest for the rest of the dissertation. We present the formal definition of semistructured data, and the current methods to storing and accessing this type of documents.

3.1 Semistructured Data

We view a semi-structured document as a labeled tree T, where each node v has a *label* $\lambda(v)$, which is a *tag name* for non-leaf nodes and a *value* for leaf nodes. Also, non-leaf nodes v have an optional set $A(v)$ of attributes, where each attribute $a \in A(v)$ has a name and a value. Note that our layout technique can also be applied to documents with cycles (e.g., ID-IDREF edges for XML documents).



root
Book [title = "XML Databases", year= 2002]
Book [title = "Storage Principles", year= 2001]
Book [title = "XML Queries", year= 2002]
Chapter [title= "XML Introduction"]
Chapter [title= "Semistructured Data"]
Chapter [title= "Implementation Issues"]
Chapter [title= "Conclusions"]
Chapter [title= "Overview"]
Chapter [title= "Introduction"]
Chapter [title= "XPath"]
Chapter [title= "Conclusions"]
Section [title= "Concurrency"]
Section [title= "Converting to XML"]
Section [title= "Hard Disks"]
Section [title= "Main Memory"]
Section [title= "Conclusions"]
Section [title= "Discussion"]
Section [title= "Open Issues"]

Figure 3.1: A sample semi-structured document.

Figure 3.1 shows an example of a semi-structured document (in this case an XML document) and Figure 3.2 shows the corresponding tree structure, created by replacing the labels with node IDs in the semi-structured tree of Figure 3.1.

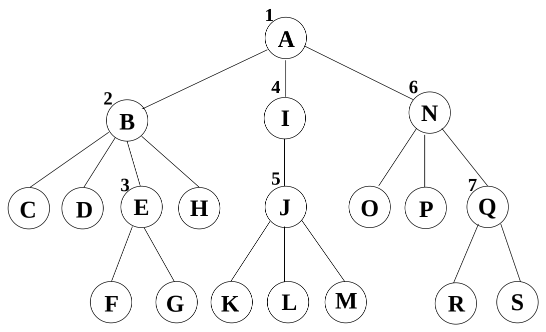


Figure 3.2: Tree structure for the XML document in Figure 3.1.

3.2 Access Model for Semistructured Data

Current-day file systems stored semi-structured data (say an XML document) sequentially on the disk. This is equivalent to placing the tree in depth-first order. To ensure a fair comparison of our storage method to the default layout, a physical pointer is added from each node to its first child and its right sibling, thereby allowing the possibility to skip the entire subtree of a node to access its right sibling. This optimization is used for the default strategy in all the experimental results we report.

For XML data, which we use as a case-study for evaluating our approach, XPath queries form the core navigation component of XML query processing systems. For evaluating XPath queries, we adopt the "standard" XPath evaluation strategy [GKP02] shown in Listing 3.1. Intuitively, this strategy processes an XPath query Q in a depth-first manner on the XML document, one step of Q ($Q.first$) at a time, and stores the intermediate results in a set S. In [BFHR06] we explain how optimizing XPath also leads to optimized XQuery.

Current implementations of XML parsers create an in-memory document tree structure that is populated (on-demand in some implementations [NSL02]) by retrieving corresponding sections of the disk-resident XML document. XML stores typically handle documents that are both smaller (i.e., tens of KB) as well as much larger size (several

Listing 3.1: Standard XPath evaluation strategy [GKP02].

```
procedure processLocationStep($n$, $Q.tail$)
  node set $S \leftarrow$ apply $Q.first$ to $n_0$;
  if $Q.tail$ not empty then
  begin
    for each node $n$ in $S$ do
      processLocationStep($n$, $Q.tail$);
    end
  end
end procedure
```

GB). Consequently, trivial solutions such as loading the entire XML document in memory prior to parsing are not deemed practical.

3.3 Disk Drive Modeling

We base our disk drive modeling on the work of [RW94b]. In their model, *seek*, *rotation*, and *transfer times*, combine the following features:

- A *seek time* that is linear with the distance, using the single-cylinder and full-stroke seek times published in the disk drive specification.
- No head-settle effects or head-switching costs.
- A *rotational delay* drawn from a uniform distribution over the interval $[0,$ *rotation time*$)$.
- A fixed controller overhead.
- A *transfer time* linear with the length of the request [RW94b].

The average random access time t_{rand}, is a function of the average seek time and rotational delay and is given by:

$$t_{rand} = seekTime\ \left(\frac{C}{3}\right) + \frac{1}{2}\,T_{rot} \qquad (3.1)$$

where $seekTime$ is a disk specific function computing the seek time given the number of tracks to seek [RW94b] and is given by:

$$seekTime(d) = \begin{cases} \alpha + \beta \cdot \sqrt{d}; if\ d < \frac{C}{3} \\ \gamma + \delta \cdot d; otherwise \end{cases} \tag{3.2}$$

where d is the seek distance in cylinders, C is the total cylinder count, and α, β, γ and δ are disk specific parameters.

CHAPTER 4

EFFICIENT STORAGE OF SEMISTRUCTURED DATA ON DISK DRIVES

4.1 Motivation

An increasing number of applications manage large amounts of semistructured data. Common applications that use semistructured data today include Bioinformatics sequence search and alignment [DKF$^+$99], genomic data analysis [Rok07], multi-resolution video storage [FJS96], clinical data systems [CDA07], XML databases, and more [PGMW95]. Given that a semi-structure such as a tree provides a more intuitive way of managing large amounts of data, the trend of storing data in such formats is likely to strengthen in the future.

Current approaches to store semistructured data either map the data to an underlying relational database system (e.g., [BFRS02b, DFS99, MH04, RFHR, STZ$^+$]), use the abstraction provided by a general-purpose object storage manager [CDF$^+$94], or use a combination of flat files and indices (e.g., XALAN [Xal07], XT [XT07], Galax [Gal07], BLAST [AGM$^+$90], Timber [JAKC$^+$02] and Natix [KM06]). Since these approaches retrofit existing storage mechanisms to work with semistructured data, their scope is restricted to the underlying mechanisms, which are predominantly optimized for sequential accesses. Consequently, these approaches may result in a mismatch between the structure and navigational primitives of semistructured data and the access characteristics of disk drives. In particular, semistructured data have a *tree* (or *graph*) structure with tree-type operations. Relational databases, on the other hand, store structured tables that are optimized for row-based access, and flat files are unstructured, optimized for sequential access. Further complicating this mismatch, the underlying storage device, *i.e.* disk drives, store information in circular tracks that are accessed with mechanical seek and rotational overhead. Given the growing amount of semistructured data, there is a need for re-examining the current storage and access machinery that support them.

In this chapter, we explore strategies to optimize the storage and retrieval of semistructured data on disk drives by explicitly accounting for the mismatch between the structure of the data and the disk drive storage and access characteristics. In particular, this chapter presents algorithms that given the physical characteristics of a disk drive (number of tracks, sectors per track and rotational speed.), place semistructured data on the disk drive in a way that facilitates navigation of the data by reducing access overheads. Such low-level control of data layout is made possible using information provided by standard disk profiling tools [WGPW95, TADP99, DRC$^+$04].

The proposed technique first addresses the problem of grouping nodes of semistructured data trees so that they can be mapped to disk blocks. This chapter presents the development and experimental evaluation of grouping strategies, which are compared with the Enhanced Kundu Misra (EKM) grouping strategy [KM06]. Second, the proposed on-disk layout strategy for node groups optimizes common tree navigation operations such as parent-to-child and node-to-next-sibling traversals. These on-disk layout strategies make use of semi-sequential disk access technique [SSS$^+$04] that allows the reduction and even elimination of rotational delay overhead during disk accesses.

Given that this approach requires circumventing the prevalent *logical block abstraction*, applying this layout strategy to a general purpose storage system is not straightforward.[1] The goal of these techniques is simply to expose the merits and demerits of this approach. Through experiments we show that our proposed approach is superior for a dedicated single-user storage system with standard caching and prefetching capabilities – for instance, a specialized system for analysis of biological data (suffix trees) [BH06].

[1]Prior research has made a similar argument in favor of fine-grained data layout by circumventing the logical block abstraction, for the case of tabular data [SSS$^+$04].

Based on this study, we believe that our approach provides a fresh perspective on the problem of storing semistructured data that is worth the attention and research time of the community.

To evaluate the proposed native data layout techniques, we used the Extensible Markup Language (XML) as a case study. XML is becoming increasingly popular due to its ability to represent arbitrary semistructured data. It is the de facto data representation format for many modern applications, including Geographic Information Systems Markup Language (GML) [GML08], Medical Markup Language (MML) [MML08], Health Level HL7 [HL708a], Clinical Document Architecture (CDA) [DAB+06] used to represent Electronic Health Records (EHRs), Open Document Format (ODF) [ODS08, oox08], and Scalable Vector Graphics (SVG) [SVG08] used to describe two-dimensional graphics and graphical applications. Despite the widespread use of XML, the challenge of optimizing access to XML data stores is a key challenge also identified in the latest report [AAB+05] on the future directions on database research, published every few years by the database research community.

Table 4.1: Query classification of popular XML benchmarks.

Benchmark	**Workload**	**Document size**	**Total queries**	**# Non-deep-focused**	**# Deep-focused**
TPoX	Financial app	2 - 25 KB	11	4	7
XMach-1	E-commerce app	2 - 100 KB	7	4	3
XMark	Auction Website	10MB - 10 GB	20	13	7
XPathMark	Education app	10MB - 10GB	54	20	34
XOO7	Web app	4MB - 1GB	23	4	19
XBench	Publications DB	1KB - 10 GB	17	11	6
MemBeR	Synthetic	11 MB	7	0	7
MBench	Synthetic	50MB - 50GB	37	37	0
Total			176	93	83

Recent surveys of popular XML benchmarks [AM06, BR03, NLB+01] show that all queries to XML data can be classified into deep-focused and non deep-focused queries. In Table 4.1, we summarize the key XML benchmarks available in the public domain. These benchmarks are further described on Section 4.7.

This collection of well-accepted and standardized XML benchmarks demonstrate:

i. that XML document sizes can be fairly large running sometimes into tens of gigabytes; this combined with the fact that XML parsers can consume as much as 5X the amount of main memory during parsing as the original size of the XML document [NJ03] implies that secondary storage accesses must be optimized if at all possible, and

ii. that the non deep-focused queries, form at least half of the total queries suggested within these popular XML benchmarks ; this implies that optimizing accesses to the non-deep-focused query class is at least as important as optimizing for the deep-focused class. Further, in the event that a workload generates both classes of queries with similar frequency, the storage system could conceivably store data using both the traditional approach and tree-based approach with the caveat that this approach requires more consideration for write-dominant workloads that can incur an unacceptable amount of overhead for maintaining consistency.

For evaluating our native layout proposals, we employ XPath queries [XPa07] obtained from the XPathMark benchmark for the evaluation. We examine the relative performance of native layout against the *default* approach, which stores XML files sequentially. To do so, we augmented an existing XML parsing engine to implement the grouping techniques that we propose. To evaluate disk I/O performance, we use an instrumented DiskSim disk simulator [BGC03] and replayed the block access traces generated by XML query processing engines. Our evaluation also addresses I/O performance in the presence of query parallelism as would be typical for server environments. Summarizing, these experiments reveal that while the default sequential layout provides superior performance for the *deep-focused* class of XML queries (or access patterns retrieving entire subtrees of semistructured data), the proposed native layout techniques outperform the default for all other query access patterns.

The rest of the chapter is organized as follows. Section 4.2 presents the architecture of a native semistructured storage system. In Section 4.3, we present native data-layout strategies for semistructured data on disk drives. In Section 4.4, we present strategies for organizing and grouping nodes in the tree so that they can be mapped to disk blocks. In Section 4.5 we conduct a theoretical analysis of the performance impact of data layout. In Section 4.6, we evaluate the proposed approach for the case of XML data by comparing it against the default sequential layout. We survey related work in Section 4.7. We conclude and discuss future directions in Section 4.8.

4.2 System Architecture

In this section, we propose an architecture for building a native semi-structured storage system which allows the use of our layout techniques with minimal changes to the current storage stack. A detailed description of the data and access model abstractions considered for this architecture can be found in Section 3.1.

4.2.1 Modifying the Storage Stack

Modern disk drives provide a high-level logical block abstraction to the operating system, which does not export information about the physical data layout, performance characteristics, and internal operation of the disk drive. We propose a modified storage stack inside the operating system that will facilitate native data layout strategies by including mechanisms to effect low-level data layout.

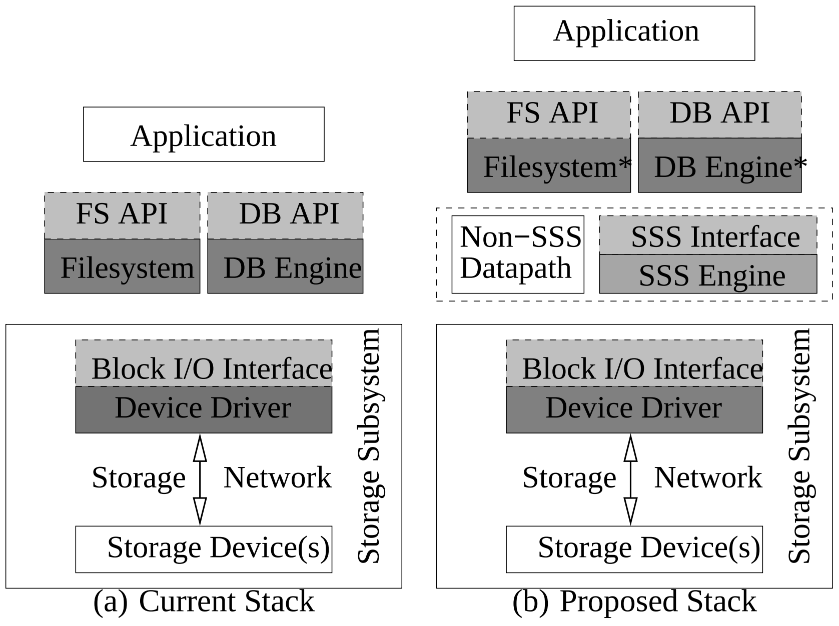


Figure 4.1: Storage stack modification.

The lowest levels of the current storage stack (shown in Figure 4.1(a)) form the storage subsystem, which exports a logical block I/O interface. The dominant storage mechanisms, i.e., databases and file systems, form the middle layer that accesses data on the storage device(s) using the logical block interface while also providing high-level APIs for applications. These storage mechanisms are optimized for relational data and sequential files respectively.

The proposed storage stack (Figure 4.1(b)) builds a native Semi-Structured Storage (SSS) engine on top of the block I/O interface to provide native storage and access support for semi-structured data. The SSS engine employs disk profiling to perform native data layout on a reserved contiguous area (partition) of the disk drive. Storage access modules (e.g. file system, database engine) need to be minimally modified to use the SSS interface in order to efficiently store and retrieve semi-structured data, or bypass it for non-semi-structured data. We chose not to build-in native support into an existing file system or existing DBMS, because we believe that the SSS engine as well as its interface can be made generic enough to work with any storage access module. Existing file and database systems can then be extended with native layout support for semi-structured data via the SSS engine. While the proposed approach call for significant changes to the operating system storage management, it is important to point out that applications retain their original interface to the operating system and remain transparent to the underlying mechanisms.

4.3 Semi-structured Data Layout

In this section, we present disk layout strategies for semi-structured data. First, we introduce a basic tree-structured placement strategy, a simple strategy which illustrates the basic ideas of our approach. Next, we present an improved and optimized variant of the basic strategy, which addresses the shortcomings of the basic strategy. Finally, we discuss some practical challenges that must be addressed when implementing the proposed placement strategies.

4.3.1 Basic Tree-structured Placement

A key limitation of the default storage method is that it is optimized only for accessing the semi-structured data tree in depth-first order since it places the data file sequentially on disk. For example, for the semistructured document in Figure 3.1 and its tree in Figure 3.2, the nodes would be stored sequentially in alphabetical order. We refer to this henceforth as the default layout and use it for comparison purposes in Section 4.6. If this file is accessed in strictly depth-first order, such a placement scheme would be optimal. However, typical tree navigation during the answering of queries displays the following characteristics: (a) nodes are accessed along any path from the root to a leaf of the tree, and (b) siblings are often accessed together, without accessing their descendants. The default layout of the nodes would result in random accesses (and therefore poor I/O performance) for both the above accesses, except for the leftmost path or traversals along leaf levels.

Based on the above observations, we design our basic layout strategy, *tree-structured placement*. To simplify the presentation of the algorithm we assume that each node in the tree occupies an entire disk block. This assumption is relaxed in Section 4.4 where we discuss in detail the grouping methods that can be employed to minimize internal fragmentation within disk blocks while maintaining the tree structure of the file.

In the basic tree-structured placement, nodes are placed on the disk starting from the outermost available track (we choose the outermost track due to its higher bandwidth, favoring the more frequently accessed higher levels of the tree). In particular, we first place the root node v on the block with the smallest logical-block-number (LBN), on the outermost available track of the disk. Second, we place its children sequentially on the next *free* track such that accessing the first child u of v after accessing v results in a *semi-sequential access* [SSS$^+$04]. This is accomplished by choosing a block for u rotationally skewed from v such that when accessing u after accessing v, the rotational delay incurred is zero. Further, accessing a non-first child from a parent node involves a semi-sequential access to reach the first child and a short rotational-delay based on the child index. The children of the first-child of the root node are then placed on the next available track, once again at a rotationally-optimal point relative to their parent. Next, the grandchildren of the first child of the root are placed following a similar approach, and so on.

As described above, the basic tree structured layout chooses parent nodes to place their respective children in depth-first order (DFO). We also experimented with breadth-first-ordering (BFO) in choosing parents, but found DFO to consistently outperform in the experiments due to its significantly shorter seek times during parent-child traversals. Intuitively, this can be visualized in Figure 3.2 where we present the DFO numbering for parent nodes (above each node); notice the localization of the numbers within each

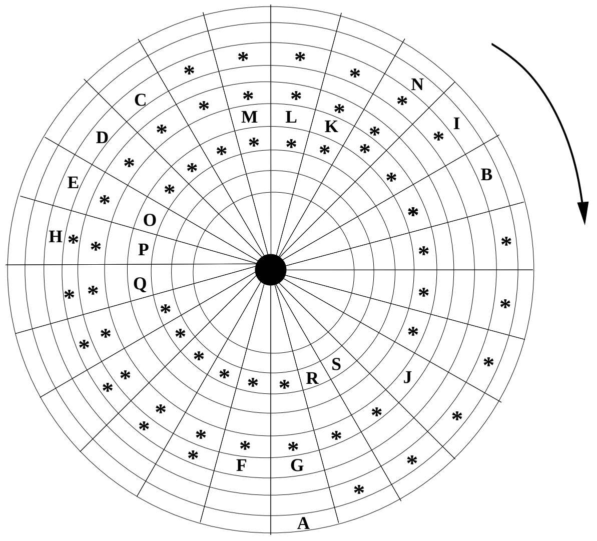


Figure 4.2: Basic tree-structured placement strategy.

subtree. The BFO ordering, on the other hand, scatters numbering over the entire tree, resulting in large seek times for parent-child traversals.

Example 4.3.1 *Figure 4.2 shows the layout of the tree of Figure 3.2 on a disk platter. To simplify presentation, we assume that the disk has a single platter with a single surface (and consequently a single disk head). Furthermore, we assume that the rotational skew between tracks is the seek-distance* $\times$ *quarter-rotation. The root node A is placed on the outermost track, track 0. Its first child B is placed on the first available free track closest to A, i.e., track 1. The block on which B is placed is rotationally skewed by a quarter-rotation relative to A as a consequence of our assumption. Accessing B after A would require only seeking to the next track. The remaining children of node A, i.e. I, and N, are placed sequentially next to the first child B. The asterisked blocks in each track immediately before the first-child represent the rotational skew between a parent and its first-child. The remaining nodes are placed following a similar approach to complete the placement of the tree.*

Listing 4.1 outlines the procedure for tree-structured placement. Notice that the leaf

Listing 4.1: Basic Placement Algorithm

```
procedure PlaceInDisk(Tree $T$)
begin
  PlaceInTrack(GetFirstFreeTrack(), 0, Root($T$))
  while there are more nodes
  begin
    $n \leftarrow$ GetNextNode()
    $t \leftarrow$ GetFirstFreeTrack()
    $L \leftarrow$ empty
    $L \leftarrow$ Add(Children($n$))
    $lbnFirstChild \leftarrow$ FindSemiSequential($n.lbn$, $T$)
    Place($t$, $lbnFirstChild$, $L$)
  end
end
```

nodes of the tree T shown in Figure 3.2 are not numbered in the ordering and hence are not returned by `getNextNode()`, which is when the placement algorithm terminates.

4.3.2 Optimized Tree-structured Placement

The basic layout strategy, as is obvious in Figure 4.2, results in severe external fragmentation of disk space (internal fragmentation within a disk block is discussed in Section 4.4), which also increases the average seek time of I/O operations. We now describe an optimization of the basic tree-structured layout strategy that reduces external fragmentation as well as random seek times drastically.

The key idea in the *optimized tree-structured placement* is the use of *non-free* tracks for placing the children for a given parent node. The optimized placement strategy is less restrictive than the basic tree-structured placement strategy in two specific ways: (1) it allows placing children on a *non-free* track, and (2) it does not require the first-child to be placed at the rotationally-optimal *block*, but rather allows placing the first-child anywhere within a rotationally-optimal *track-region* as defined next.

We define a *track-region* as a contiguous list of N_{tr} disk-blocks along a track. The blocks within a track-region, therefore, are also sequential in the logical address space (LBN space) of the disk. Given a parent node u and a target track t, we define the *rotationally-optimal track-region* for u on track t as the track-region of size N_{tr} blocks starting from the block where the disk head lands when seeking to track t starting from u. In Figure 4.3, two rotationally-optimal track-regions (N_{tr}=6) for parent node 'S' are marked using the $\#$ symbol. To place the children nodes for node u, a set of *candidate* rotationally-optimal track-regions are chosen close to u, which can lie on either side of the parent track. The optimized placement algorithm chooses the track-region closest to u with sufficient free space to house the children of u. Other than this variation, the optimized tree-structured placement algorithm proceeds to place the tree similar to the basic placement algorithm.

In the above placement description, the choice of the rotationally-optimal track-region size (N_{tr}) is a critical factor. Increasing the track-region size gives the placement algorithm more opportunity to reduce fragmentation and consequently reduce random-seek overhead between node accesses, but it also increases the average rotational delay incurred during parent-to-child node-traversals. This is an important trade-off to be considered when choosing N_{tr}. In our experiments, we choose N_{tr} as a quarter of the track-size.

Figure 4.3 shows the layout of the tree in Figure 3.2 on a hard disk (platter) using the optimized strategy. Again, we assume that the platter rotates in the clockwise direction. The assumptions of track skew are also the same as for the basic strategy. In the optimized placement, since a single track can contain the children of several nodes, the external fragmentation (shown in Section 4.6) is drastically reduced compared to the basic tree-structured placement.

The `PlaceInTrack` method in Listing 4.2 outlines the logic for optimized tree-structured placement. Line 1 places the root node of the tree T on the outermost track.

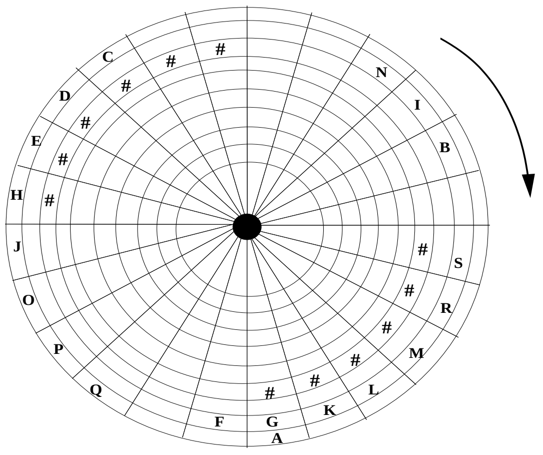


Figure 4.3: Optimized Strategy.

Lines 2-7 place the children of the *next* node (which is the root node in the first iteration) on the rotationally-optimal track-region (returned by `FindRotTrackRegion`). The next node is returned by `getNextNode()`, which returns a non-leaf node of the XML tree based on the chosen ordering scheme. The above process is repeated until all the nodes are placed on the disk. The auxiliary method `GetTrack(LBN)` returns the track for LBN; the auxiliary method `FreeTrackRegionStart(LBN, int, tracks)` recieves as parameters a parent LBN, its number of children, and the number of tracks to skip, and returns the LBN for the first child if all children can be placed in the candidate tracks rotationally-optimal track-region. Otherwise returns NULL. Candidate tracks are the two tracks situated at parentTrack +/- tracksToSkip respectively.

Notice that the leaf nodes of T are not numbered in the ordering and hence are not returned by `getNextNode()`. The `findRotTrackRegion(LBN parent,int nchildren)` auxiliary method checks for availability of space in the rotationally optimal track-regions in tracks on either side of the parent's track, starting from the closest track. It returns the LBN for placing the first-child of the `parent` node. The remaining children are placed incrementally following the first child. The `direction` identifier specifies where the target track lies with respect to the parent. If the `direction` has

Listing 4.2: Optimized Placement Algorithm

```
procedure <Track, LBN> FindRotTrackReg(LBN $parent$, int $n$)
begin
  $tracksToSkip \leftarrow 1$
  $parentTrack \leftarrow$ GetTrack($parent$)
  while true
  begin
    $lbnFirstChild \leftarrow$ FreeTrackRegionStart($parent$, $n$,
        $tracksToSkip$)
    if $lbnFirstChild$ not NULL
    begin
      return <GetTrack($lbnFirstChild$), $lbnFirstChild >$
    end
  end
  $tracksToSkip++$
end

procedure PlaceInDisk(Tree $T$)
begin
  PlaceInTrack(getFirstFreeTrack(), $0$, root($tree$))
  while there are more nodes
  begin
    $n \leftarrow$ GetNextNode()
    $L \leftarrow$ empty
    $L \leftarrow$ add(children($n$))
    $< lbnFirstChild > \leftarrow$ FindRotTrackReg($n.lbn$, $L$.size())
    Place($target$, $lbnFirstChild$, $L$)
  end
end
```

a negative value, the target track is less than the parent track. Likewise, a positive value indicates that the target track is greater than the parent track.

4.3.3 Implementation Issues

In implementing the strategies presented above, several practical issues must be considered. First, the above placement scheme assumes that a single, contiguous partition, large enough to accommodate the semi-structured data is available. This assumption is realistic for both file systems and database systems since they typically allocate a large contiguous disk partition and can reserve a fraction of this space for storing semi-structured data.

Second, after a tree node is read from the disk drive, a non-negligible CPU think time is typically required before the next I/O request is issued. We address this issue as follows. If the next request is for a sibling node (stored sequentially in our approach), then on-disk pre-fetching mechanisms ensure that this node is pre-fetched into the on-disk cache. However, if the next request is for a child node (stored semi-sequentially), then during computation time, the disk would have already rotated by an amount proportional to the CPU think time and hence no semi-sequential access would be possible. To address this, we skew the first child by an additional rotational delay equivalent to 95^{th} percentile of a sample from the think time distribution. This ensures that in most cases, the semi-sequential nature of child node accesses will be preserved.

Third, the proposed strategy would work well when processing a single query at a time. However, if there are multiple queries issued concurrently by different processes or users, then the resulting interleaving I/Os are likely to degrade sequential or semi-sequential accesses to random ones. This problem is prominent even in traditional relational database and filesystem accesses. Techniques at the disk scheduling layer such as *anticipatory scheduling* [ID01], which group together requests from a single process and minimize the effects of multiple interleaved I/O request streams, address this issue well. We evaluate the impact of query parallelism (in Section 4.6) with anticipatory I/O scheduling to demonstrate the effectiveness of native layout strategies in the simulated environment.

Finally, existing storage interfaces are restrictive which makes it non-trivial to obtain profiling information or control data layout. While the need for more expressive storage interfaces has been brought up repeatedly in the storage research community(e.g., [Gan01, KPH98, RGF98]), for the time-being, we can circumvent this restriction by employing disk profiling and control tools. Profiled information includes: rotational time, seek time, track and cylinder skew times, sizes of read cache and write buffer along with pre-fetching and buffering techniques, logical to physical block mappings, and access time prediction. This profiled information enable fine-grained control for disk drives, tailored specifically for semi-structured data.

4.4 Supernode Trees

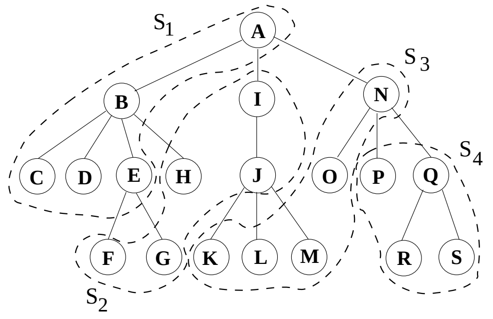
So far, we assumed that each node in the semi-structured data tree occupies an entire disk block. This assumption, however, is not realistic; in practice, the tree nodes are of variable size, ranging from a fraction of a disk block to multiple disk blocks.

In this section, we first lay the foundation for grouping nodes in a semi-structured data tree T to form *supernodes* where each supernode occupies an entire disk block. Next, we describe how to organize the supernodes into a supernode tree structure T_S. The placement strategies of Section 4.3 are then applied on the supernode tree instead of the node tree.

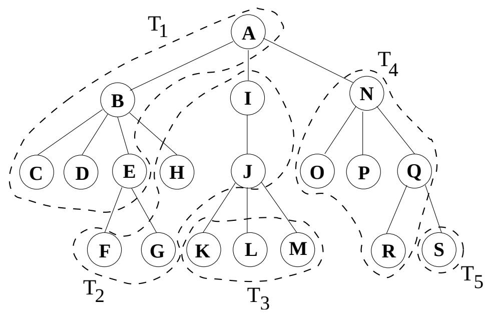
4.4.1 Grouping Nodes into Supernodes

To reduce the internal fragmentation, it is desirable to group the maximum number of nodes into a supernode. It is also important to group adjacent nodes of T in the same supernode, so that navigating among these nodes requires only one disk access. If the size of a node is larger than the size of a disk block, it is stored using multiple supernodes, which are then stored in consecutive disk blocks. [2]

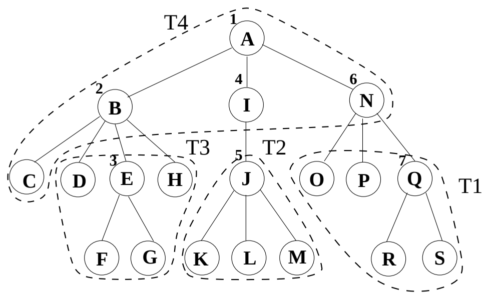
To elucidate the following grouping techniques, we assume that all nodes have the same size, and one supernode can contain at most five nodes.



(a) Sequential grouping



(b) Tree-preserving grouping



(c) EKM grouping as described in [KM06]

Figure 4.4: Grouping strategies for creating supernodes.

[2]An alternative strategy to avoid breaking the tree-structure of the rest nodes would be to store a pointer to a Binary Large Object (BLOB) and use an object storage manager [CDF$^+$94] to manage BLOBs.

Sequential grouping. Nodes are added to a supernode starting from the root node using a depth-first (and left-to-right) traversal. The only difference is that a single node is not split nodes across disk blocks, unless the size of the node is greater than the size of a disk block. Figure 4.4(a) illustrates this grouping strategy for the tree presented earlier in Figure 3.2.

Tree-preserving grouping. The tree-preserving grouping proceeds as in the sequential grouping except it ensures that cycles of supernodes do not form in the grouped tree. At each step, before adding a node v to a supernode S, the following additional conditions are checked:

(i) the parent node of v is in S, or

(ii) the parent node of v is in the parent supernode of S.

If any of these conditions hold, then we add v to S. If neither holds, then by adding v to S a cycle of supernodes in the original tree T would be created. To avoid that, we close S and add v to a new supernode. This strategy aims at preserving the tree-structure of the original tree T in the supernode tree. Figure 4.4(b) illustrates this grouping strategy for the tree of Figure 3.2.

Enhanced Kundu Misra grouping. We also implement a grouping technique developed independently at the same time by Kanne and Moerkotte [KM06] called the Enhanced Kundu Misra (EKM) grouping, an extension to the original Kundu-Misra grouping algorithm [KM77]. The EKM strategy operates in a bottom-up fashion and aims at reducing the number of node groups while preserving the original tree structure, thereby increasing navigations between nodes within the same group. It operates by converting the n-ary tree into a binary tree representation, obtaining a layered partitioning that helps reducing the number of supernodes while preserving the connectedness. Figure 4.4(c) illustrates this grouping strategy for the tree of Figure 3.2.

4.4.2 Building Supernode Trees

The organization of the supernodes into a supernode tree, T_S, determines the placement of the supernodes on the disk drive according to the algorithms presented in Section 4.3. Hence, it is desirable to preserve the tree-structure of T in T_S. That is, if a parent-child pair of nodes in T is split to different supernodes, then it is preferable to split it to two adjacent supernodes in T_S. Based on the grouping strategies described above, we consider four supernode tree organization strategies:

1. The *sequential supernode list*, which corresponds to the default placement strategy, uses sequential grouping to form supernodes. It is merely a linked-list of supernodes in the order in which the supernodes were formed. Figure 4.5(a) shows the formation of this list.

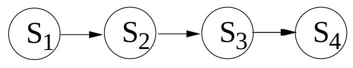
2. The *tree-preserving supernode tree*, which corresponds to the *tree-preserving*[3] *tree-structured*[4] *placement* to be introduced in Section 4.6, uses the tree-preserving grouping to form supernodes. The supernode tree is formed by adding edges between two supernodes S_i, S_j if there is an edge between two nodes $v_i \in S_i, v_j \in S_j$ in T. Notice that due to the nature of tree-preserving grouping no cycles can occur. Figure 4.5(b) shows the formation of this tree.

3. The *sequential supernode tree*, which corresponds to the *sequential tree-structured placement algorithm* in Section 4.6, uses the sequential grouping to form supernodes. Then, the supernode tree is created by adding edges between pairs of supernodes S_i, S_j if there is an edge between two nodes $v_i \in S_i, v_j \in S_j$ in T and adding the edge will not create a cycle. Figure 4.5(c) shows the formation of this tree.

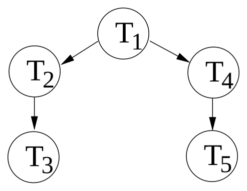
4. The *EKM supernode tree* builds a tree on the EKM supernodes. Again no cycles exist due to the nature of EKM grouping. Figure 4.5(d) shows the formation of this tree.

[3]with respect to grouping

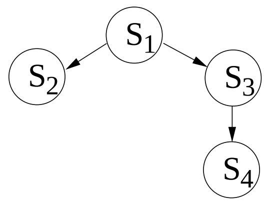
[4]with respect to placement algorithm



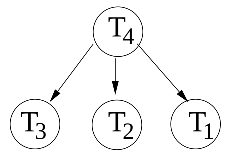
(a) Sequential supernode list.



(b) Tree-preserving supernode tree.



(c) Sequential supernode tree.



(d) EKM supernode tree.

Figure 4.5: Supernode Trees.

4.5 Theoretical Analysis

In this section, we present a quantitative model to analyze the access times for the default and the optimized tree-structured placement strategies. Table 4.2 summarizes the description of each parameter used in this analysis.

First we compute the random, sequential and semi-sequential access times, following the equations and models described in Section 3.3. For the barracuda disk, chosen as the base disk configuration in the experiments (and also further described in Table 4.7), the rotational latency is given by $T_{rot} = 8.33$ ms and $\alpha = 1.83, \beta = 0.17, \gamma = 2.85$ and $\delta = 0.0035$. For an XML document of size 50MB occupies 129188 blocks or 325 cylinders after grouping with the tree-preserving grouping strategy (Table 4.4). Thus, substituting these values in the above Equation 3.1, the random access time for the area occupied by this document is given by $t_{rand} = 5.99$ ms.

The average sequential access time t_{seq} from one block to the next is a very small

Table 4.2: Disk Drive Parameter Description

$T_{default}$: Average access time in default placement T_{tree}: Average access time in tree-structured placement
t_{seq}: Average access time for sequential access t_{rand}: Average access time for random access $t_{semi-seq}$: Average access time for semi-sequential access
a_1: Access is from parent to first child a_2: Access is from a parent node to non-first child a_3: Access is from a non-leaf node to its right sibling a_4: Access is from a leaf node to its right sibling a_5: All other accesses (that is, $P_5 = (1 - (\sum_{i=1}^{4} P_i))$
P_i: Probability that access a_i occurs; $1 \leq i \leq 5$
$t_{default}(a_i)$: Average time for a_i in default placement $t_{tree}(a_i)$: Average time for a_i in tree-structured placement
C: Number of Cylinders
T_{rot}: Rotational Period
T_{nt}: Time taken to transfer one block of data

value, approaching zero. Hence,

$$t_{seq} = 0 \tag{4.1}$$

For the tree-structured placement, the access between a parent and its first child is semi-sequential, and from a node to its right sibling is sequential. The average time for semi-sequential access $t_{semi-seq}$ given by:

$$t_{semi-seq}(v) = seekTime\ (s(v)) \tag{4.2}$$

where $s(v)$ is the number of tracks to be seeked during a semi-sequential access. When T is a complete tree with height d and degree f, the average $s(v)$ is given by:

$$s(v) = \frac{f^{d-2}(d - 2 - f/(1-f)) + 2 + f/(1-f)}{2n'} \tag{4.3}$$

where n' is the number of internal nodes given by $n' = \frac{(1-f^{d-1})}{(1-f)}$

To understand this equation, let's assume that the root is at depth 1 and the leaves at depth d. If there are two edges $u_1 - v_1$ and $u_2 - v_2$ where u_1 and u_2 are on the same level and v_1 and v_2 are their l^{th} respectively, then $DFO(v_1) - DFO(u_1) = DFO(v_2) - DFO(u_2)$. Thus, the distance in tracks from v_1 to its child u_1 and from v_2 to u_2 are the same. In the above relation, $DFO(x)$ is the corresponding number in the DFO ordering. The numbers above the internal nodes in the tree shown in Figure 3.2 illustrate the DFO ordering.

To calculate the average $s(v)$ for the nodes v of level $k+1$, we need to find the size of the subtree rooted at v which is

$$1 + f + \cdots + f^{d-k-1} = \frac{(1 - f^{d-k})}{(1-f)} \tag{4.4}$$

The average of $s(v)$ for the nodes v of level $k+1$ is the average $s(v)$ of any set of siblings at level $k+1$. That is,

$$\frac{\left(\frac{f+(1-f^{d-k})}{(1-f)(1+\cdots+(f-1))}\right)}{f} = \frac{\left(\frac{f+(1-f^{d-k})}{(1-f)(f-1)f/2}\right)}{f} = \frac{(f^{d-k}+1)}{2} \tag{4.5}$$

Hence, for level k it is $\frac{(f^{d-k-1}+1)}{2}$.

For an average fanout of 10 and a depth of 5 in an XML tree, $s(v)$ from Equation 4.3 is 1.83. Thus, the $seekTime(s(v))$ is $\alpha + \beta \cdot \sqrt{1.83} = 2.26$.

Equation 4.2 assumes perfect semi-sequential time, which is achieved by the tree-structured algorithm (Algorithm 4.1). However, in the case of the optimized tree-structured algorithm (Algorithm 4.2), $t_{semi-seq}(v)$ depends on the number of track-regions per-track, k. Hence,

$$t_{semi-seq}(v) = seekTime\ (s(v)) + \frac{1}{2k} T_{rot} \tag{4.6}$$

Since the first-child is placed anywhere within a rotationally-optimal track-region rather than rotationally optimal sector, accessing the first child may involve anywhere between 0 to $\frac{1}{k}T_{rot}$ rotational delay after the seek operation. This additional rotational delay during the semi-sequential access is $\frac{1}{2k}T_{rot}$ on an average. When a track is divided in 8 track regions, k =8 and for the barracuda disk, $s(v)$ is calculated above and is 1.83

ms. Substituting these values in Equation 4.6, the average semi-sequential time is given by $t_{semi-seq}(v) = 2.79$ ms, a significant reduction of 53.4 % from an average random access time of 5.99 ms.

Next, we discuss the time needed for each of the five basic access types of Table 4.2. When the first child is accessed from its parent (a_1), a sequential access occurs in the default placement, whereas a semi-sequential access occurs in the tree-structured placement. When a non-first child is read from its parent (a_2), it is a random access in the default placement, whereas for the tree-structured placement, it is the sum of the semi-sequential time and the average sibling index ($f/2$, where f is the tree fanout) times T_{nt} (time required to transfer data from one node). When the access is from a non-leaf node to its right sibling (a_3) it is a random access in the default placement, and a sequential access in the tree-structured placement. When from a leaf-node we access its right sibling (a_4), it is a sequential access in either placement strategy. In all other cases (a_5), such as when moving up the tree, for both placements a random access will be performed. Table 4.3 summarizes the access times in the default and the tree-structured storage for every a_i.

Table 4.3: Average access times in default and tree-structured placement for each access type a_i.

Access type a_i	Description	$t_{default}(a_i)$	$t_{tree}(a_i)$
a_1	Parent to first child	t_{seq}	$t_{semi-seq}$
a_2	Parent to non-first child	t_{rand}	$t_{semi-seq} + \frac{f}{2}(T_{nt})$
a_3	Non-leaf node to right sibling	t_{rand}	t_{seq}
a_4	Leaf node to right sibling	t_{seq}	t_{seq}
a_5	All other accesses	t_{rand}	t_{rand}

The average access times in default and tree-structured storage are computed by Equations 4.7 and 4.8 respectively.

$$T_{default} = \sum_{i=1}^{5} P_i \cdot t_{default}(a_i) \tag{4.7}$$

$$T_{tree} = \sum_{i=1}^{5} P_i \cdot t_{tree}(a_i) \tag{4.8}$$

Tree-structured placement is better when $T_{tree} < T_{default}$.

While this is not realistic (and necessarily subjective to the query as demonstrated extensively later in Table 4.6), if we did assume that a query exhibits all the access types shown in Table 4.3, with each access type occurring equally frequently, the average I/O times for the default and the tree placement can be obtained by substituting their values in Equations 4.7 and 4.8 as:

$$T_{default} = \frac{1}{5} \cdot t_{seq} + \frac{1}{5} \cdot t_{rand} + \frac{1}{5} \cdot t_{rand} + \frac{1}{5} \cdot t_{seq} + \frac{1}{5} \cdot t_{rand}$$

= 3.594 ms, and

$$T_{tree} = \frac{1}{5} \cdot t_{semi-seq} + \frac{1}{5} \cdot (t_{semi-seq} + \frac{f}{2}(T_{nt})) + \frac{1}{5} \cdot t_{seq} + \frac{1}{5} \cdot t_{seq} + \frac{1}{5} \cdot t_{rand}$$

= 2.344 ms

where the transfer time T_{nt} = 0.03 ms.

4.6 Evaluation Case Study: EXtensible Markup Language (XML)

In this section, we experimentally evaluate the grouping and native layout strategies for placing XML data on disk drives.

We used the DiskSim [BGC03] disk simulator for our evaluations, instrumenting it to provide the additional interface:

```
<LBN> findSemiSequential( LBN parent, int cyl, int track )
```

which given a parent LBN, returns an LBN X on <cyl,track>, such that access from the parent LBN to X is semi-sequential.

The optimized-tree placement in Algorithm 4.2 uses this interface to find semi-sequential LBA for subsequent nodes in the tree that has to be placed on the disk. The optimized tree-structured and the default placement algorithms were implemented in C and integrated with the instrumented DiskSim code. The grouping algorithms were implemented as a separate module.

4.6.1 Data Set and Queries

We generated XML files (each file corresponds to an XML tree) of various sizes using the XMark generator [SWK$^+$02b] with different scaling factors from $f = 0.01$ to $f = 1.00$, corresponding to file sizes ranging from 1MB to 100MB. The limit of 100MB for the maximum file size is due to the memory constraints in currently available open-source XML parsing engine implementations. These engines create the navigation tree data structures for the entire tree in memory during parsing, while at the same time consuming as much memory as five times the original document size [NJ03].

Earlier in Table 4.1, we presented the document sizes used by several popular benchmarks typically used to evaluate XML query optimizations, storage, indexing and so on. As mentioned earlier in Section 4.2, trivial solutions that load the entire document in memory are not practical for large (several gigabyte sized) XML documents. Although the XML documents we experiment with are small relative to the size of the disk, these serve as examples to illustrate the *relative* effectiveness of native layout when compared to the existing approaches. It should additionally be noted that the on-disk buffer is small (1-8MB) for the disks we use, substantially smaller relative to the size of the documents, and is not in any significant way capable of influencing the I/O access patterns apart from on-disk readahead.

We implemented the three grouping strategies – *sequential, tree-preserving,* and *EKM* – described in Section 4.4, computing and storing the information about the supernode that would contain each XML node. We also implemented extensions to the DiskSim

Table 4.4: XML Tree and Supernode Tree Parameters

XMark factor	Tree (MB)	#Nodes x1000	B/node (Avg)	# Supernodes x1000			B/supernode (Avg)		
				TP	Seq.	EKM	TP	Seq.	EKM
0.01	1.7	17.1	25.2	2.5	2.1	2.1	343.8	418	412.3
0.05	8.3	59.6	25.8	12.8	10.6	10.7	373.2	450.8	447.5
0.10	16.8	167.8	25.8	26.0	21.4	21.6	345.3	418.7	414.9
0.50	83.7	832.9	26.1	129.2	106.6	114.8	345.3	418.5	414.6
1.00	168.7	1666.3	26.1	259.6	214.3	216.1	345.3	418.2	414.7

disk simulator [BGC03] that allowed us to simulate the native layout strategy described in Section 4.3. We then used the supernode information to store them on disks simulated by DiskSim.

Table 4.4 provides information about the XML trees used and the corresponding supernode trees formed. The number of supernodes in the sequential grouping is the lowest since it groups the nodes to form supernodes without any restrictions. EKM does a bottom-up grouping of the tree and reduces the number of resulting supernodes by reducing the problem of finding supernodes for arbitrary trees to the simpler problem of finding supernodes for flat trees (trees in which all nodes but the root are leaves) [KM06]. Tree-preserving grouping avoids cycles by placing restrictions on the nodes being added to the supernode. This in turn reduces the number of nodes per supernode and subsequently increases the number of supernodes. The average nodes/supernode is six for the tree-preserving grouping and is 8 for Sequential and EKM grouping.

For the query workload, we adopted performance-sensitive queries from the XPathMark benchmark [Fra04], but omitted the ones that check for features supported by XPath (e.g., *Q18: /comment()*). To compute reliable results we added more queries with similar properties of depth, number of conditions and selectivity. The query workload is summarized in Table 4.5.

To contrast the relative advantages of using our native strategies with those of the

Table 4.5: XPath queries for the deep-focused (D) and the non deep-focused (N) classes.

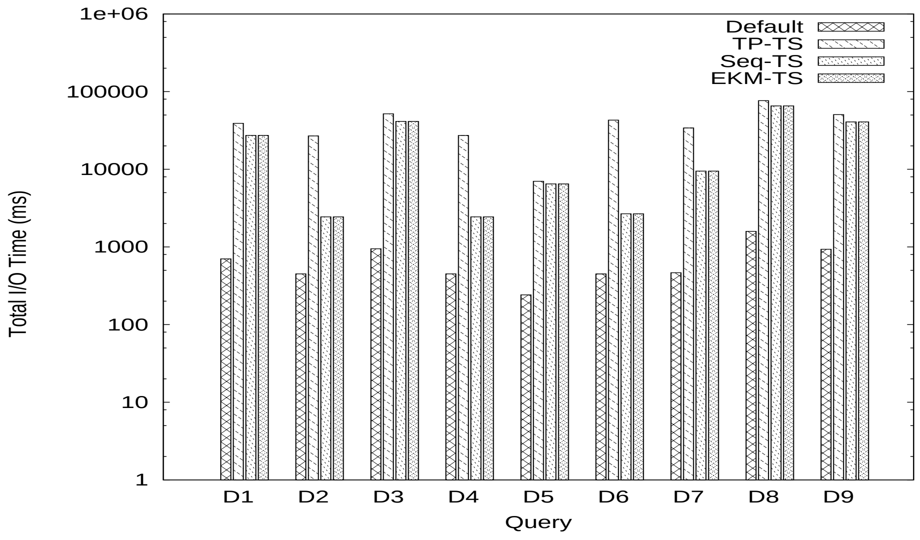
#	Deep-focused Query
D_1	$/site/closed\_auctions/closed\_auction/annotation/description/parlist/listitem/text/keyword$
D_2	$/site/people/person/watches$
D_3	$/site/open\_auctions/open\_auction/annotation/description/text/keyword$
D_4	$/site/people/person/address/country$
D_5	$/site/regions/australia/item/description/text/emph$
D_6	$/site/people/person/*/business$
D_7	$/site/closed\_auctions/closed\_auction/*/description$
D_8	$/site/regions/*/item/description/text$
D_9	$/site/closed\_auctions//itemref$
#	Non deep-focused Query
N_1	$/site/open\_auctions/open\_auction$
N_2	$/site/closed\_auctions$
N_3	$/site/regions/australia$
N_4	$/site/closed\_auctions/closed\_auction$
N_5	$/site/regions/*/item$
N_6	$/site/*/australia$
N_7	$/site/open\_auctions/open\_auction[@id =' open\_auction0']/bidder$
N_8	$/site/regions/asia/item[@id =' item4']/mailbox/mail/from$
N_9	$/site/open\_auctions/open\_auction[@id = "open\_auction0"]//keyword$

default sequential layout, we classify XPath queries into two categories: *deep-focused queries* and *non deep-focused queries*. A subset of each class is shown in Table 4.5. The former class describes the special class of XPath queries that navigate entire subtrees of the tree (queries $D_1, \ldots, D_9$ in Table 4.5). The latter class, non deep-focused queries $N_1, \ldots, N_9$ in Table 4.5, represents all queries that do not belong to the former class. As we shall demonstrate, the default layout primarily addresses the class of deep-focused queries and is sub-optimal for all other queries. Notice that only the supernode-granularity navigation matters for overall I/O performance, and not the node-granularity navigation. Hence, queries like D_2, which do not access leaf nodes, are included in the first category since they access supernode leaves; the *watches* subtree is very small and fits in less than one supernode.

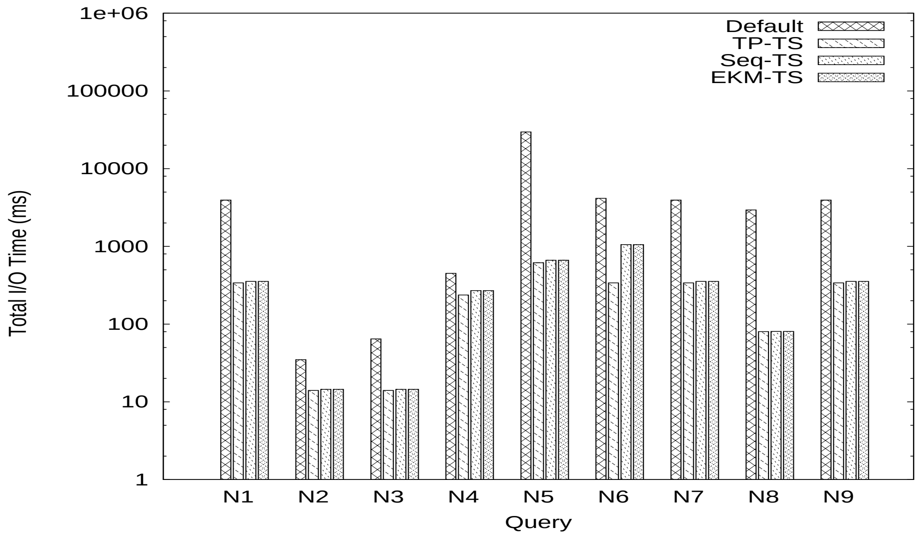
4.6.2 Tree Navigation Performance

We conducted experiments that compare the I/O times for answering XML queries for four different layout strategies, corresponding to the supernode tree organizations of Section 4.4: *default* (Section 3.1), *tree-preserving tree-structured* (TP-TS), *sequential tree-structured* (Seq-TS), and *EKM tree-structured* (EKM-TS) layout strategy.

To consider caching effects in our experiments, we assumed that all nodes along the path from the root to a single leaf node would be cached in main memory, either in the operating system VFS or a custom application level cache. This is a reasonable assumption for XML trees, which are typically short even when their total size is large, due to large fan-out. Consequently, we ignore repeated accesses to nodes (such as parent, ancestor nodes) during the depth first traversal of the XML tree. Such caching reduces the number of random accesses equally in all three placement strategies, since the navigation of nodes for answering a query is exactly the same regardless of the layout strategy.

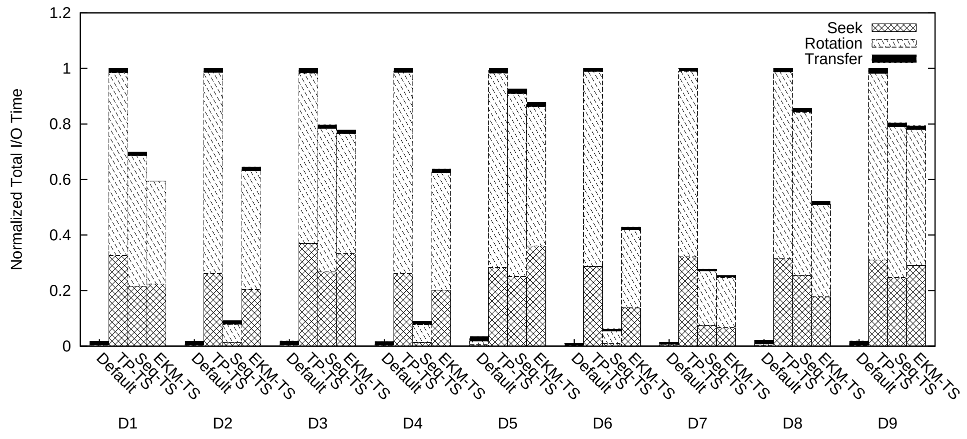


(a) Deep-focused queries

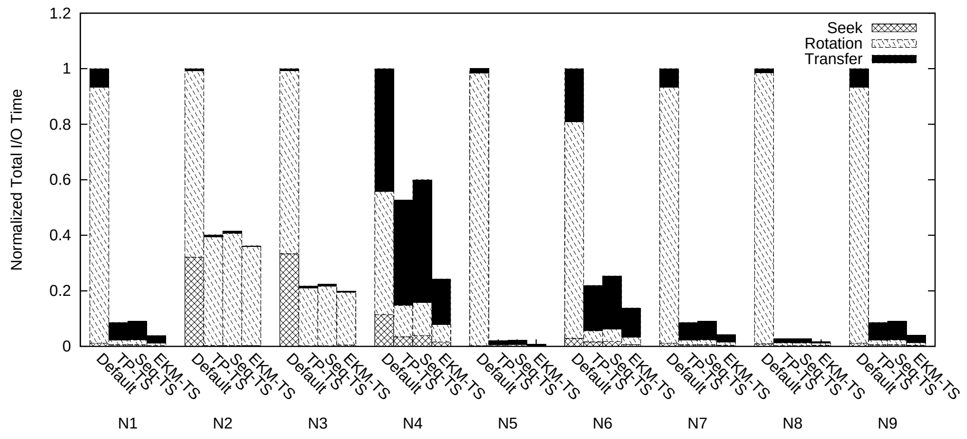


(b) Non-deep-focused queries

Figure 4.6: Total I/O times in logarithmic scale for various placement strategies.



Disk
(a) Deep-focused queries



Disk
(b) Non-deep-focused queries

Figure 4.7: Normalized total I/O times for various placement strategies.

Table 4.6: Navigational patterns for the two XPath query classes for f = 0.5. a_i's are defined in Table 4.2.

Default Placement											
Query	a_1	a_2	a_3	a_4	a_5	*Query*	a_1	a_2	a_3	a_4	a_5
D_1	9046	0	0	0	1982	N_1	1098	0	0	0	4775
D_2	7211	0	0	0	55	N_2	0	0	0	0	5
D_3	12744	0	0	0	1895	N_3	0	0	0	0	10
D_4	7211	0	0	0	55	N_4	1387	0	0	0	3053
D_5	1823	0	0	0	759	N_5	1322	0	0	0	9323
D_6	7315	0	0	0	4	N_6	9324	0	0	0	8418
D_7	2765	0	0	0	2814	N_7	1098	0	0	0	4775
D_8	11937	0	0	0	9654	N_8	121	0	0	0	870
D_9	16166	0	0	0	5	N_9	1098	0	0	0	4775
TP-TS Placement											
Query	a_1	a_2	a_3	a_4	a_5	*Query*	a_1	a_2	a_3	a_4	a_5
D_1	4438	1182	1799	1114	5117	N_1	1	1	71	5513	1
D_2	3250	3	333	1801	3251	N_2	0	1	0	4	0
D_3	6171	1729	2428	902	7897	N_3	0	1	0	9	0
D_4	3287	3	333	1764	3288	N_4	0	2	42	3762	0
D_5	659	319	507	169	976	N_5	0	6	42	10065	5
D_6	5218	1	371	3	5049	N_6	4	2	485	14647	4
D_7	1344	2665	42	71	3758	N_7	1	1	71	5513	1
D_8	4071	4831	1360	2164	8896	N_8	0	2	2	937	1
D_9	8213	1	4657	4	7199	N_9	1	1	71	5513	1
Seq-TS Placement											
Query	a_1	a_2	a_3	a_4	a_5	*Query*	a_1	a_2	a_3	a_4	a_5
D_1	6856	1073	1768	219	1112	N_1	1074	859	5	24	3911
D_2	6714	47	47	0	458	N_2	0	1	0	0	4
D_3	9582	458	2347	123	2129	N_3	0	1	0	0	9
D_4	6714	47	47	0	458	N_4	1347	777	2	7	2307
D_5	1149	175	487	33	738	N_5	1305	2576	0	103	6661
D_6	6765	1	95	0	458	N_6	8771	1719	83	47	7122
D_7	2620	1098	2	44	1815	N_7	1074	859	5	24	3911
D_8	9193	3364	1385	715	6934	N_8	120	227	0	6	638
D_9	10564	1	4602	0	1004	N_9	1074	859	5	24	3911
EKM-TS Placement											
Query	a_1	a_2	a_3	a_4	a_5	*Query*	a_1	a_2	a_3	a_4	a_5
D_1	2126	4153	1795	1319	5521	N_1	0	2	88	2305	1
D_2	2040	1117	3342	1983	3156	N_2	0	1	0	0	0
D_3	3259	5042	3838	731	7981	N_3	0	1	0	4	0
D_4	2040	1117	3342	1983	3156	N_4	0	2	151	1495	0
D_5	445	1106	395	287	1414	N_5	0	6	89	3588	5
D_6	2242	1129	3347	1924	3306	N_6	0	6	3584	6174	4
D_7	803	2000	151	1237	2801	N_7	1	2	88	2304	2
D_8	2730	9672	913	3323	12399	N_8	0	2	12	327	1
D_9	3180	2581	6116	0	4029	N_9	1	2	88	2304	1

Total I/O time

Figure 4.6 shows (in logarithmic scale) the I/O times for each query, for the two classes of queries, deep-focused (D_i) and non deep-focused (N_i), for an XMark file with scaling factor $f = 0.5$. We executed five simulation runs for each column shown in the graph. For the first run, the start LBA for the placement of the root node was 0. For all the subsequent runs, it varied with increments of 250 ($>$ track size). Thus, the start LBA was varied over the range $0 - 1250$. The confidence interval, for a confidence level of 95%, for all the five runs was found to be $< \pm$ 10.96. The results shown in the graph are for the start LBA 0.

For the deep-focused class of queries, the default placement strategy performs consistently better than the others, since it can retrieve entire subtrees more efficiently. For the non-deep-focused query class, the performance of the default placement strategy is consistently worse than the tree-structured variants (TP-TS, Seq-TS, and EKM-TS). For this query-class, a large number of accesses are non-sequential for the default placement, since complete sub-tree accesses are few.

Figure 4.7 shows the relative performance with the normalized total I/O time to reduce the impact of the large variance across queries. Each value is scaled relative to the maximum value for the experiment. To better demonstrate the relative distribution of seek, rotational delay, and transfer time components, the total normalized I/O time is further split to show these I/O access time components. It can be seen that the average rotational delays for the tree-structured placement strategies (in the case of non-deep-focused queries) are substantially lower relative to the default strategy. However, this is not the case for the deep-focused class where the default strategy outperforms in all respects.

To better understand and explain the graphs of Figure 4.6 and Figure 4.7, we counted the different types of accesses in the supernode tree (each access translates to a disk I/O operation) for answering the XPath queries for both the deep-focused and non deep-focused classes. Table 4.6 shows the numbers of supernodes accesses for the five basic types of tree accesses, a_1 through a_5, defined in Table 4.3. As an example, observe that for the TP-TS placement, Query D_1 requires 4438 a_1 accesses, the parent-to-first-child type accesses.

We can make some general observations from Table 4.6. First, the default placement causes all the accesses to be either of type a_1 or a_5, since only parent-to-first-child sequential accesses are possible for this layout. Second, the deep-focused queries are dominated by a_1 and a_5 type accesses, while the non-deep-focused queries are dominated by a_3 and a_4 accesses (except in the case of default placement). This enables the non-deep-focused queries to exploit native layout, since all the accesses to siblings are sequential, as opposed to the large number of random accesses the deep-focused queries require. Observe further that the EKM and TP-TS placement strategies increase the number of accesses from parent to non-first child, thus utilizing the semi-sequential and sequential access optimization to a larger extent. For the deep-focused queries, on the other hand, the default placement erforms the best both because the number of sequential accesses for his placement is the highest and number of random accesses is lowest (in most cases) among all placement techniques.

In Figure 4.7 (b), we see a somewhat unexpected outcome that the seek times reduce for queries N_2 and N_3 for TP-TS, Seq-TS and EKM placement. An answer can be found in the access patterns of these queries (Table 4.6). For N_2 and N_3, all accesses for the default placement are of type a_5, which are random accesses, where as for the TP-TS and EKM placement, they are either semi-sequential or sequential accesses, leading to the observed difference in seek overhead. Further, the Seq-TS has a slightly lower performance relative to these two because of the increase in the number of random accesses for this placement. Note that although the number of random accesses in Seq-TS is relatively higher, it is still lower than the default placement and hence it performs better than the default placement.

The above discussion serves to reinforce the arguments we made earlier when discussing Figure 4.6. In summary, the EKM-TS placement strategy performs better overall due to its lower internal fragmentation and tree-structure preservation property; it results in I/O times which are 3X-127X better than the default strategy. Between the remaining strategies, TP-TS performs better on an average, since it better preserves the original tree-structure.

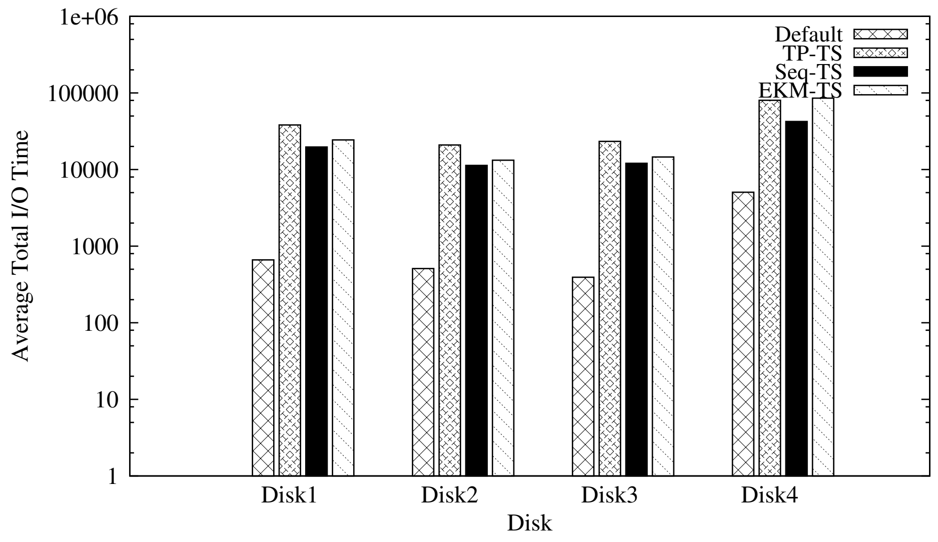
Sensitivity to drive characteristics

To evaluate the effect of drive characteristics, we conducted a sensitivity study of I/O access time for representative disk-drive models. The drive models chosen, shown in Table 4.7, were the Seagate Barracuda, Seagate Cheetah 9LP, Seagate Cheetah 4LP, and the HP C3323A as representative of four performance classes of disk drives: *base, fast rotating and fast seeking, fast rotating,* and *slow rotating* respectively. A disk block is of size 512 bytes.

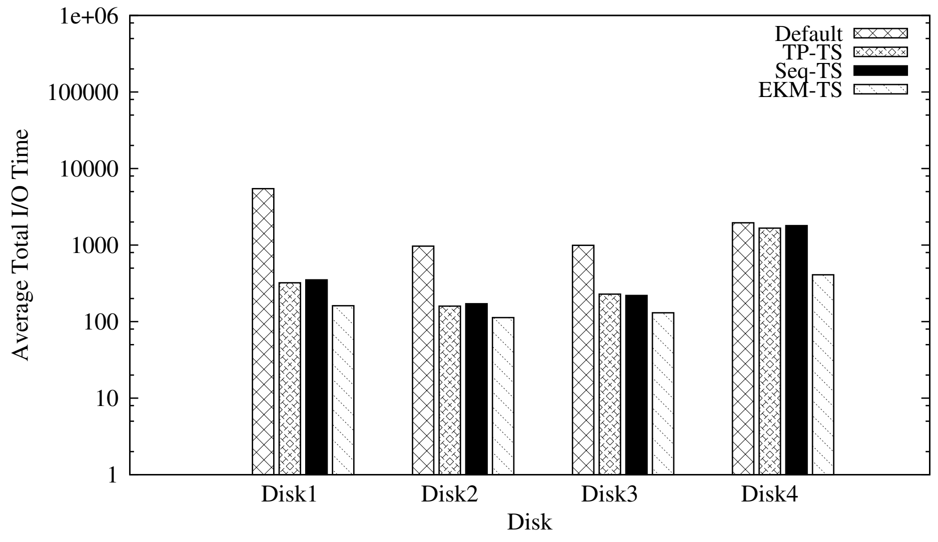
Table 4.7: Characteristics of experimented disk drives.

Disk model	**Disk type**	**Size [GB]**	**RPM**	**Stroke [ms]**	**Transfer [MBps]**	**Sectors / track**	**Cylinders**
Barracuda	Base	2.0	7200	16.679	10-15	119-186	5172
Cheetah 9LP	Fast disk	9.1	10045	10.627	19-28.9	167-254	6962
Cheetah 4LP	Fast rot.	4.5	10033	16.107	15-22.1	131-195	6581
HP C3323A	Slow rot.	1.0	5400	18.11	4.0-6.6	72-120	2982

Figure 4.8 shows the average (across queries in a query-class) total I/O times (in logarithmic scale) for the two query classes for an XMark file with $f = 0.5$ with the various hard disk models. For the special class of deep-focused queries (Figure 4.8(a)), the default placement strategy performs better than the other strategies benefiting from

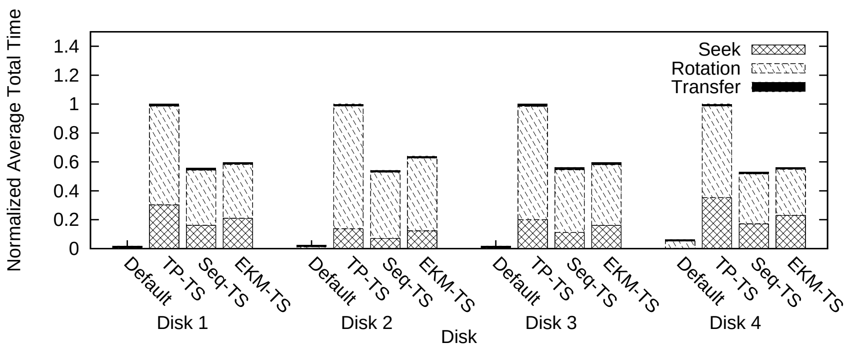


(a) Deep-focused queries

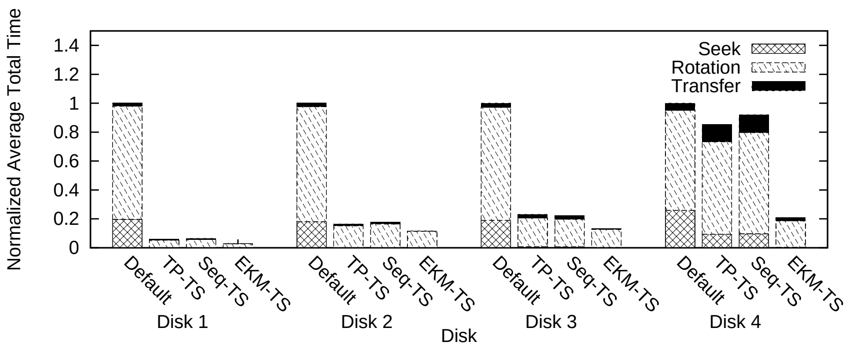


(b) Non-deep-focused queries

Figure 4.8: Sensitivity of query I/O times to changing disk drive characteristics (logarithmic scale).



(a) Deep-focused queries



(b) Non-deep-focused queries

Figure 4.9: Sensitivity of seek and rotational delay components of I/O access times to changing disk drive characteristics.

optimized sub-tree retrievals. However, for all other queries (Figure 4.8(b)), the tree-structured placement strategies perform better for all disk models, offering as much as 7X-34X reduction in average I/O time for answering queries. This underscores the importance of native layout strategies for XML data.

We break down the gains further in Figure 4.9 into the relative reduction in seek and rotational delay components for each of the drives by normalizing the I/O times at each disk drive using the maximum value as reference.. Notice for the non-deep-focused query class (Figure 4.9(b)), the average rotational-delays are substantially reduced relative to the default layout.

Effect of Query Interleaving

One concern with a native layout targeted to a optimize a specific access pattern is the impact of multi-processing in the system. For instance, a server is likely to execute multiple XPath queries simultaneously; optimizing individual query executions may not necessary translate to overall performance improvement when the corresponding I/O request sequences are interleaved. As elaborated in Section 4.3, this issue in its more general form (i.e., multi-process blocking I/O performance) has been addressed earlier with anticipatory I/O scheduling [ID01]. Consequently, we expect that XML servers would be configured with I/O schedulers that include an anticipation core.

To evaluate the performance of our grouping and placement techniques under multiple simultaneous XPath queries, we interleaved a subset of deep-focused and non-deep-focused queries stated in Table 4.5. The interleaved queries belonged to either the disjoint set of queries which accessed disparate portions of the tree or intersecting queries whose access paths overlapped. The ordering of the I/Os after interleaving were based on anticipatory scheduling. We simulate the behavior of the anticipatory I/O scheduler assuming that each query is serviced within an independent thread and issues synchronous I/O requests. The behavior of the non-work-conserving anticipatory scheduler would result in optimizing the schedule of successive I/O operations resulting from the same query, in spite of them being issued synchronously, as long as other queries in the system access disjoint portions of the XML tree. When there is an overlap of subtrees between two queries, their I/Os must interleave.

Table 4.8: Query Interleaving for Multi-User Simulations.

Disjoint Queries	**Deep-focused Queries**	**Non-deep-focused Queries**
δ_1	$D_1 + D_4$	$N_1 + N_4$
δ_2	$D_4 + D_8$	$N_4 + N_8$
δ_3	$D_5 + D_7$	$N_5 + N_7$
δ_4	$D_1 + D_4 + D_5$	$N_1 + N_4 + N_5$
δ_5	$D_4 + D_5 + D_7$	$N_4 + N_5 + N_7$
δ_6	$D_4 + D_5 + D_9$	$N_4 + N_5 + N_9$
Intersecting Queries	**Deep-focused Queries**	**Non-deep-focused Queries**
π_1	$D_1 + D_7$	$N_1 + N_6$
π_2	$D_2 + D_4$	$N_5 + N_6$
π_3	$D_5 + D_8$	$N_7 + N_9$
π_4	$D_4 + D_6$	$N_1 + N_6 + N_7$
π_5	$D_1 + D_7 + D_9$	$N_6 + N_7 + N_9$
π_6	$D_2 + D_4 + D_6$	$N_5 + N_6 + N_8$

For the choice of queries, we selected both *disjoint queries*, which traverse different subtrees of the document, as well as *intersecting queries*, that access common subtrees, which navigate common sub-trees of the document. Table 4.8 shows the selected queries that were interleaved in each of these categories, where δ_i refers to disjoint queries and π_i represents intersecting queries.

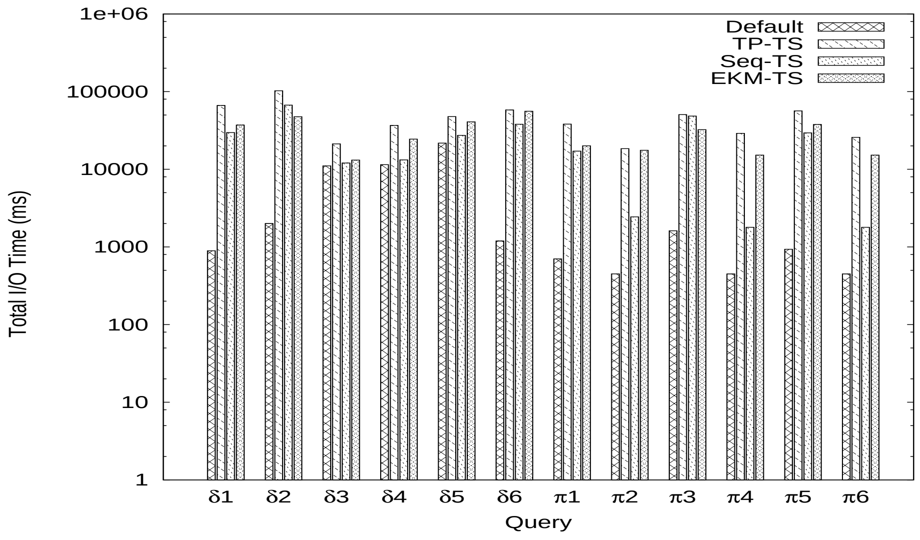
Figure 4.10 shows the total I/O time (in logarithmic scale) for the execution of interleaved deep-focused and non-deep-focused XPath queries. The results for the deep-focused queries from Figure 4.10 (a), show that like in single query execution, the default strategy performs better for multiple interleaved queries than the other strategies.

Similarly, the behavior with the the non-deep-focused interleaved queries mostly mimic their single query counterparts. The native layout strategies provide much better execution times for both the disjoint and intersecting queries, as shown in Figure 4.10 (b). Moreover, the EKM-TS performs better the most consistently across the interleaved query executions. The breadth-first grouping approach of this placement strategy causes the I/Os corresponding to the upper levels of the XML tree to be read in parallel. For lower tree levels, the anticipatory scheduler which ensures that the I/O sequences generated by the individual query threads are grouped successfully. Finally, the default placement performs consistently worse for the disjoint queries, since the I/O sequences generated by individual query threads are executed almost sequentially.

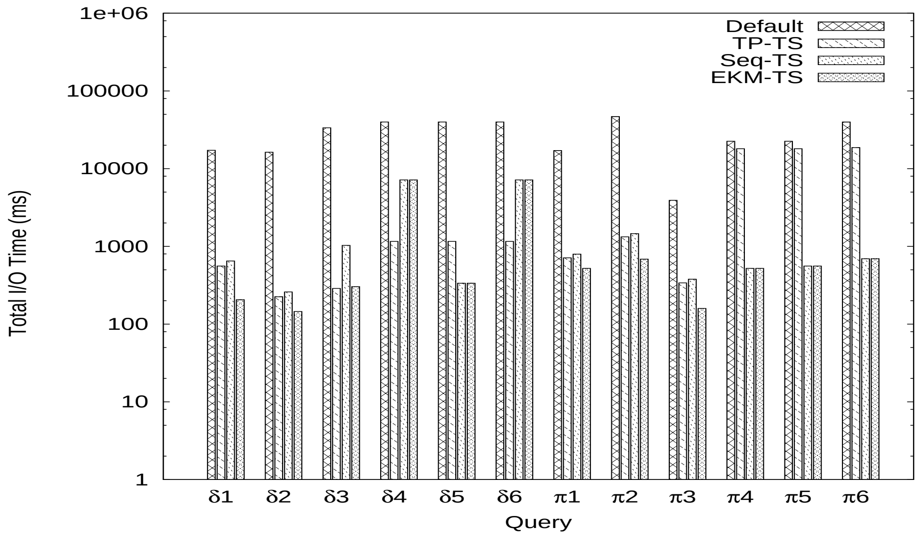
4.6.3 Fragmentation

We now measure the internal and external fragmentation incurred by the grouping and placement algorithms respectively.

Internal Fragmentation: Figure 4.11 (a) shows the internal fragmentation of disk block space with the three grouping algorithms, *sequential*, *tree-preserving*, and *EKM*. As expected, the sequential grouping algorithm has little internal fragmentation as it can freely add nodes to a supernode as long as adding the next node does not violate the block-size restriction. Supernodes are not occupied completely if its the remaining space is smaller than the size of the next XML node. The tree-preserving grouping places further restrictions on grouping for preserving the XML tree-structure in supernodes and incurs additional internal fragmentation (as much as 55%). We argue that considering the fact that current disk drives are bound more by I/O access time than by I/O capacity, trading

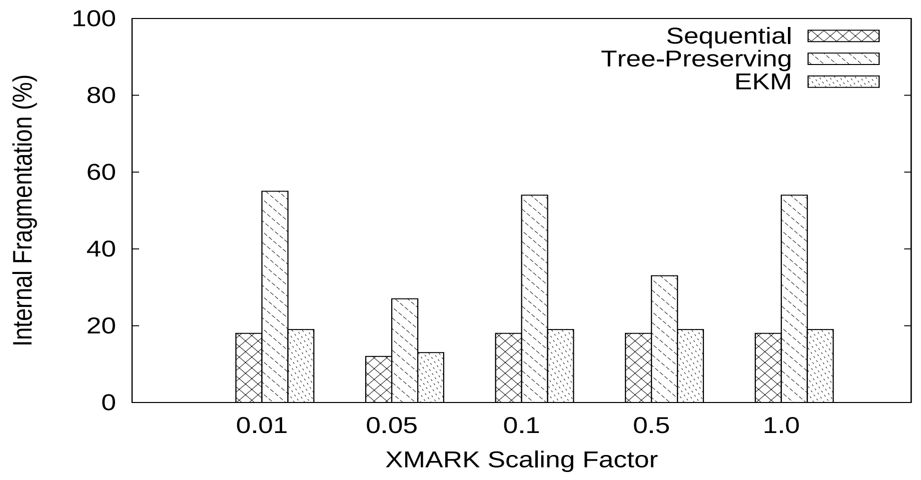


(a) Deep-focused queries

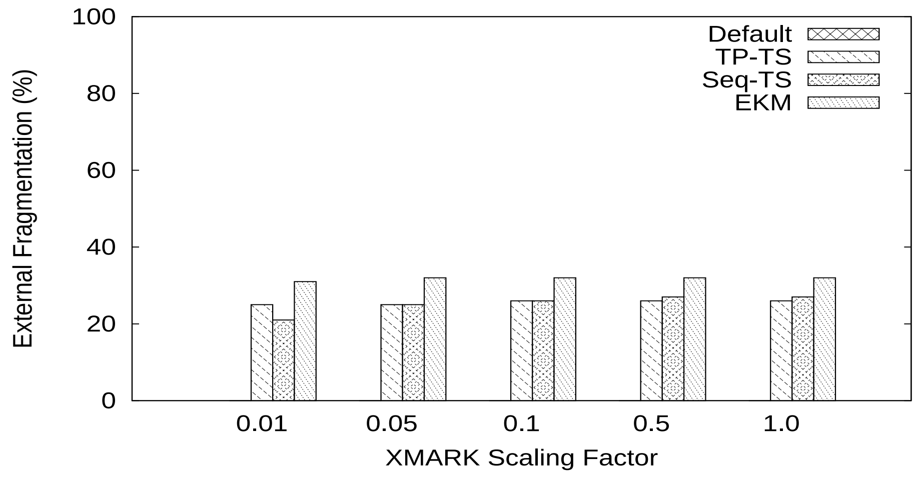


(b) Non-deep-focused queries

Figure 4.10: Total I/O times in logarithmic scale for interleaved XPath queries.



(a) Internal fragmentation



(b) External fragmentation

Figure 4.11: Internal and External Fragmentation.

capacity for improving access is acceptable. The internal fragmentation with EKM is very close to that for sequential grouping. The EKM algorithm has the flexibility that allows selecting any of a node's many subtrees as partition, thereby obtaining a more optimal result for this procedure. Our tree-preserving grouping algorithms lack this flexibility, and can only add the next node to the current supernode in an in-order fashion.

External Fragmentation: Figure 4.11 (b) shows the external fragmentation results for the data placement strategies. The default strategy incurs zero external fragmentation as it places the supernode list sequentially on the disk. TP-TS and Seq-TS incur external fragmentation of less than 28%, while that of the EKM-TS is higher at around 32%. However, we once again contend that these numbers are acceptable, following the arguments mentioned above. EKM-TS incurs the highest external fragmentation, because in EKM-TS, the fanout of nodes is less in the top levels (closest to root) of the tree and is higher in the lower levels, unlike the other strategies. If the fanout of a tree is higher at a greater depth, it is more difficult to find contiguous free space to place all the children on the partially occupied tracks using the optimized placement strategy. Consequently the children are placed on new tracks, thereby increasing the external fragmentation. Furthermore, for a native storage solution that is well integrated into the existing file or database system, it is relatively easy to utilize fragmented free space.

4.7 Related Work on Storing Semistructured Data

Storage of semi-structured data has received attention in the last few years because of its growing popularity. Most work has focused on storing semi-structured data in relational DBMSs or in flat files with indexes. The former approach (e.g.,[BBM$^+$01, DAYF, STZ$^+$, NNP00, DFS99, MAG$^+$97]) has been the most popular due to the success and maturity of the relational DBMSs. The latter approach (e.g., [KBNK02, LM01]) is based on storing the data as a flat file and building separate indexes on top. These strategies do not use native layout of semi-structured data and are limited to the generic optimization strategies built into relational databases and file systems.

The problem of native storage of semi-structured data has been addressed in Natix [KM99, KBM05] and in System RX [BCJ$^+$05], where the tree-structured data is split into pages and each page is stored in a disk block, thereby reducing the number of read accesses while traversing the tree. OrientStore [MLLA03] uses schema information to make a storage plan for the semi-structured data. The above studies however view a disk drive as a list of pages and do not take into account the physical characteristics of its operation whereas we investigate how to exploit detailed information about the disk drive and use this information to minimize overheads such as seek-time and rotational-delay.

Given the restrictive block IO interface, the clear case for a more expressive interface has been made before [Gan01]. Systems such as [GNA$^+$, HSW$^+$04, SPP$^+$03] use intelligence from upper layers of the storage stack inside storage devices to improve overall IO performance. Our work, if deployed, can use such systems, to incorporate storage techniques for semi-structured data into disk firmware.

Recent work by [SSP$^+$05] uses the idea of semi-sequential access for efficient storage of multi-dimensional data. This work is significantly different from our work in that unlike semi-structured data, multi-dimensional data is structured with access patterns along data dimensions and can afford efficient layout based on fixed attribute cardinality. Also, with semi-structured data, grouping multiple data elements to be stored on a disk block is non-trivial due to the variable size of the data elements.

Atropos [SSS$^+$04] exploits the physical properties of disk drives and uses semi-sequential accesses to store relational databases. Our work targets XML data that has a tree structure, quite different from the relational tables. We also show that a naive application of the semi-sequential access paradigm to XML tree structures leads to large seek times and severe space fragmentation. Our optimized layout strategy reduces such overhead significantly. To the best of our knowledge, there is no existing work tackling the problem of laying out XML data, accounting for low-level hard drive storage and operation semantics.

XML Benchmarks: The Transaction Processing over XML (TPoX) benchmark [NKS07] evaluates the performance of XML stores, XML databases, and indexes, by generating a mix of XQueries for various financial transactions on the generated XML documents. XMach-1 [BR01, BR03], XOO7 [BDL$^+$], XMark [SWK$^+$02a] and XPathMark [Fra04] are typically used to evaluate query optimizations in XML. XMach-1 is based on an E-commerce website while XMark generates queries for an E-commerce website with information on bids, items, brokers and customers. XPathMark [Fra04] is an XPath based benchmark for XMark and generates an educational document that represents the English alphabet. The XBench [YOK03] benchmark is an application oriented benchmark for XML databases. Finally, the Michigan (MBench) [RPJ$^+$03] and the MemBer [AMM05, MMM06] Benchmarks are both micro-benchmarks that generate synthetic workloads wherein document structure can be finely controlled (varying their depth and fan-out) so as to be able to reproduce the access patterns of a variety of different real-world workloads.

4.8 Conclusions

In this chapter, we have taken a first step towards building native storage systems for semi-structured data, a problem which has been largely unexplored. We presented on-disk data layout techniques for semi-structured data that explicitly account for the structural mismatch between the semi-structured data and disk drives and reduce disk access overhead. These layout techniques are based on node-grouping algorithms for semi-structured data that reduce the number of disk I/O operations required when accessing the data. We have suggested directions for addressing the challenges that would arise in integrating the proposed layout techniques in existing storage systems.

Summary of Experimental Findings and Lessons Learned

We conducted an evaluation of the native layout techniques using XML as a case-study. All experiments were performed on XPathMark benchmark queries with an instrumented DiskSim simulator. Our experiments revealed that:

- For the specific class of *deep-focused* queries, which result in access patterns retrieving entire sub-trees, the existing file system layout mechanism (i.e., sequential layout of the tree in depth-first-order) offers significantly better performance than native layout (5X-54X across the query set). For such queries, we believe that sequential layout is the right choice.
- For all other query classes, which we group as *non-deep-focused*, native layout taking into account tree navigation primitives, offers as much as 3X-127X performance improvement across the range of XPathMark queries that we experimented with, representing a large improvement. A sensitivity study across a range of disk models, representing drives of varying performance, suggest that average I/O performance improvement across the non-deep-focused query set of 7X-34X.
- Of the various native layout techniques we considered, the EKM-TS provided consistently better performance, barring a few cases. The above findings were largely preserved when we experimented with multiple simultaneous query executions with the anticipatory I/O scheduler. This scheduler naturally carries forward the benefits of native layout into the I/O schedule.
- Native layout strategies, however, can result in substantial fragmentation of disk space. Our initial estimates reveal total fragmentation (internal+external) of as

much as 50% for the best-performing EKM-TS layout technique. This fragmented space can be reclaimed with clever file system or database system implementations to store non semi-structured data. Even if that were not feasible, we believe an additional 50% of space overhead for several magnitudes of I/O bandwidth increase could be acceptable in many settings.

Our findings in this study serve to more closely examine and evaluate layout techniques based on the nature and distribution of queries (i.e., access patterns). Further, based on our findings in this study, it can be inferred that a single layout technique is unlikely to be optimal for navigating semi-structured data; the optimality of any layout technique closely depends on the nature of the workload. A prudent choice of the underlying data layout strategy can drastically improve I/O access times if knowledge of the access patterns (e.g., query workload) is available beforehand.

CHAPTER 5

EFFICIENT PARSING OF SEMISTRUCTURED DOCUMENTS

5.1 Motivation

XML has become the de facto standard format for data representation and exchange in domains ranging from the Web to desktop applications. Examples of XML-based document types include Geographic Information Systems Markup Language (GML) [GML08], Medical Markup Language (MML) [MML08], HL7 [HL708a], and Open Document Format (ODF) [ope08, oox08]. This widespread use of XML requires efficient parsing techniques. The importance of efficient XML parsing methods was underscored by Nicola and John [NJ03]; they showed that the parsing process is processor and memory consuming, particularly needing main memory as much as five times the size of the original document.

There are two popular XML parsing APIs, DOM [DOM08] and SAX [sax08]. SAX reads the whole document and generates a sequence of events according to the nesting of the elements, and hence it is not possible to skip reading parts of the document as this would change the semantics of the API. On the other hand, DOM allows users to explicitly navigate in the XML document using methods like `getFirstChild()`, `getNextSibling()`, and so on. DOM is the most popular interface to traverse XML documents because of its ease of use. Unfortunately, its implementation is inefficient since entire subtrees cannot be skipped when a method like `getNextSibling()` is invoked. This also leads to frequent "Out of memory" exceptions. In contrast to SAX, parsing a document using DOM could potentially avoid reading the whole document as the sequence of navigation methods may only request to access a small subset of the document. In this work we focus on parsing using a DOM-like interface.

Lazy XML parsing has been proposed (e.g., [xer08]) to improve the performance of the parsing process by avoiding the loading of unnecessary elements. This approach substitutes the traditional eager evaluation with a lazy evaluation as used by functional programming languages [Abr90]. The architecture shown in Figure 5.1, based on the terminology of [NSL02], consists of two stages. First, a preprocessing stage extracts a virtual document tree, which stores only node types, hierarchical structure information and references to the data associated with each node. After this structure is obtained, a progressive parsing engine refines this virtual tree on demand, which grows as needed, expanding the original virtual nodes into complete nodes with values, attributes, and textual information.

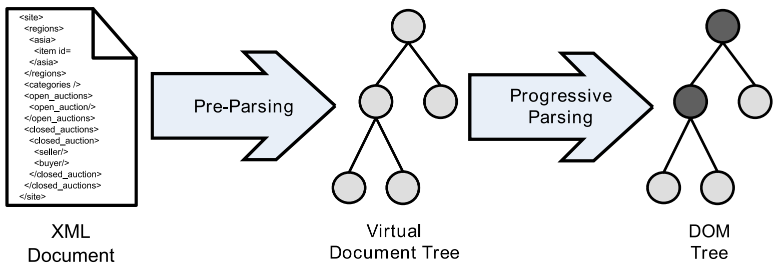

```
<site>
  <regions>
    <asia>
      <item id=
    </asia>
  </regions>
  <categories />
  <open_auctions>
    <open_auction/>
  </open_auctions>
  <closed_auctions>
    <closed_auction>
      <seller/>
      <buyer/>
    </closed_auction>
  </closed_auctions>
</site>
```

Figure 5.1: Lazy XML Parser Architecture. A pre-parsing phase extracts a virtual document tree and a progressive parsing engine refines this virtual tree on demand.

Clearly, the lazy parsing technique is a significant improvement. However, it still suffers from the high initial cost of pre-parsing (Figure 5.1) where the whole document must be read before the lazy/progressive parsing starts. The pre-parsing stage is inevitable due to the lack of internal physical pointers (or something equivalent) within the XML document. We propose a method to (a) insert such internal physical pointers in the document, and (b) exploit them to optimize the parsing method and specially the pre-parsing stage. In particular, our approach is called *double-Lazy Parsing (2LP)* because both stages in Figure 5.1 are lazy, in contrast to previous work where only the second stage is lazy. The pre-parsing phase will lazily process only the subtrees of the XML document that are necessary to satisfy the navigation request.

We address two key issues in inserting such physical pointers. First, we need to decide how we can implement the pointers given the current W3C XML standard specification [BPSM$^+$06]. Second, we need to decide where to add the pointers, considering the incurred overhead adding pointers on every node can cause the size of the file to double. Also, following a pointer would typically require a random disk access, and hence excessive use of such pointers must be avoided.

Regarding the first issue, we emulate physical pointers, by partitioning the original XML document into several fragments (subtrees) which are then interlinked using the XML Inclusion [xin08] feature. A drawback of this approach is that the XML document is split into a set of smaller XML documents/files[1]. However, we shall argue and demonstrate in the rest of this chapter that the performance gains far outweigh this drawback. Regarding the second issue, we investigate in detail the tradeoff decisions to be made with respect to fragment size, and propose an optimal configuration that can be applied in general cases.

We also propose a method to manage the parsing of large XML documents under limited main memory configurations. This approach allows 2LP to scale to large XML documents, even when the total size of the document surpasses the available amount main memory. This is not possible with current parsers, which report “Out of memory” exceptions under such condition.

This chapter makes the following contributions:

1. We develop a framework to allow efficient XML parsing, which improves the pre-

[1]Unfortunately, the XML standard does not support an alternative physical pointer construct (XPointer [xpo08] is logical and not physical) due to the complication this would incur during cross-platform document exchange. If such a feature becomes available in the future, it could be used instead of the described partitioning approach.

parsing time as well as the memory requirements of parsing. Our framework is based on the idea of placing internal physical pointers within the document. Such pointers are currently realized using the XML Inclusion feature.

2. We present algorithms to perform double-Lazy XML Parsing (2LP) for DOM-like navigation, given internal physical pointers. We have implemented 2LP as a backward compatible modification of the Apache Xerces2 Java Parser [xer08].

3. We present algorithms to add internal physical pointer to the XML document by partitioning it into subtrees given an optimal partition size. We show how the theoretically optimal partition size can be computed assuming knowledge of the navigation patterns on complete XML trees and knowing the hard disk characteristics.

4. We efficiently manage the main memory consumption of our XML parser, making it possible to parse and navigate large documents under conditions in which other approaches fail.

5. We study our partitioning and parsing algorithms both theoretically and experimentally. Experiments on various XML navigation patterns, including XPath, confirm our theoretical results and show consistent and often dramatic improvement in the parsing times.

The rest of this chapter is organized as follows: Section 5.2 presents the system framework and the overview of our approach. We describe our double-Lazy parsing techniques in Section 5.3. Section 5.4 presents techniques for partitioning the original document into smaller subtrees. An approach to parse using a limited amount of main memory is presented in Section 5.5. The implementation of all these techniques is discussed in Section 5.6. Our experiments are discussed in Section 5.7. We present related work in Section 5.9. Finally, Section 5.8 discusses our conclusions.

5.2 System Framework and Overview of Approach

In this section we present the data and query models and the global overview of our approach.

5.2.1 Data and Query Models

XML data: We view an XML document as described in Section 3.1. For simplicity in the presentation we assume that there are no ID-IDREF edges (which would make the tree a graph). However, our framework can support ID-IDREF edges by including the partition id, in addition to the attribute id, in the IDREF attributes. Figure 5.2 shows a sample XML tree, extracted in a similar way to Figures 3.1 and 3.2. We annotate each node with its size and the size of its subtree (in parenthesis). To simplify the discussions in the rest of this chapter, we assume that these sizes are in numbers of disk blocks. Similarly, the number in the parenthesis represents the size (in blocks again) of the subtree rooted at the node.

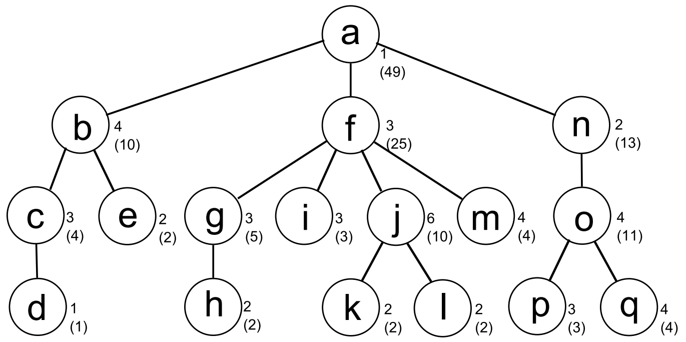


Figure 5.2: Sample XML tree. We annotate each node with its size and the size of its subtree (in parenthesis).

We clarify that this work is not aiming at improving the performance of XML database systems [Gal07, Xal07, XT07, JAKC$^+$02, Nat06], where indexes [GW97, Gru02] and other optimizations are possible, but at improving the efficiency of using XML as a format to store documents for general applications as motivated in Section 5.1.

XML navigation patterns: We consider two types of XML navigation patterns in our experiments. The first type is a simple *root-to-leaf traversal*, in which a path is traversed from the root of the XML document to any of its leaves. We use this simple yet common and useful pattern to model the theoretical behavior of our approach.

Second, we use XPath queries. We use XPath and not XQuery because our work tackles the problem of efficient parsing for the purpose of efficiently navigating the XML data, which is XPath's role. However, our results for XPath carry to XQuery as well, since XQuery queries are typically evaluated by combining the results of the involved XPath queries. Again, we adopt the "standard" XPath evaluation strategy [GKP02], as shown in Algorithm 3.1 in Section 3.2.

5.2.2 Disk Drive Modeling

We utilize the disk drive characteristics and models presented in Section 3.3 to obtain the transfer time and random access time for the set of hard disk drives that use for our theoretical model and experimental section. Table 5.1 presents the disk drive *transfer time* (t_{transf}) and *random access time* (t_{rand}), required to transfer and access a disk block respectively, for the four hard drive disks we utilize for our theoretical model and experimental section. The values presented in this table were gathered from the manufacturers data sheets [Hit09, Max09, Qua09, Sea09].

Notice that according to their model definition, the typical seek times are the average *seek*, *track-to-track seek*, and *full stroke*. We also consider the analysis of the average seek distance, utilizing one third of the full stroke as the average distance seek.

Table 5.1: Hard Drive Modeling Parameters

Disk Model	**Maxtor 6L020J1 [Max09]**	**Quantum Fireball+ KX27.3 [Qua09]**	**Seagate Cheetah 15K.4 [Sea09]**	**Hitachi UltraStar 10K300 [Hit09]**
Formatted capacity (GB)	20.0	27.3	36.7	73.4
Heads	1	16	2	3
Rotational Speed (RPM)	7200	7200	15000	10025
Stroke (ms)	17.8	15	7.9	10
Transfer (MBps)	54.2	66.6	200	134.375
Block count	40,132,503	54,600,000	71,687,372	143,374,804
Cylinders	16 383	16 383	50 864	65 494
Avg. seek	8.5	8.5	3.5	4.3
Track switch	0.8	0.8	0.2	0.4
Full Stroke	17.8	15	7.9	10

The equations in Table 5.2 describe the Gamma function that models the head positioning effects as stated in [RW94b], approximating the measured seek-time profile for the different disk drives.

Table 5.2: Gamma Function: Seek Curve Modeling

seek distance	**γ (ms)**
$< 1/3$ Cylinders	$a + b \cdot \sqrt{distance}$
$\geq 1/3$ Cylinders	$c + d \cdot distance$

As stated in Table 5.2 the average seek distance will be less than one third of the cylinders, we use the first equation to calculate γ. Table 5.3 summarizes the values for the four parameters a, b, c and d, as well as the Gamma function value and the final Transfer Time and Random Access Time.

Table 5.3: Gamma Values, Transfer and Random-access Time

Disk Model	**Maxtor 6L020J1**	**Quantum Fireball+ KX27.3**	**Seagate Cheetah 15K.4**	**Hitachi UltraStar 10K300**
a	0.694374	0.694374	0.174460	0.373425
b	0.105626	0.105626	0.025540	0.026575
c	3.850000	5.250000	1.300000	1.450000
d	0.000851	0.000595	0.000130	0.000131
$\gamma(1/3\, cylinders)$	1.275209	1.192346	0.359615	0.528011
t_{transf}	0.009446	0.007688	0.002560	0.003810
t_{rand}	5.441876	5.359013	2.359615	3.520530

5.2.3 Overview of Approach

Our approach for parsing XML documents consists of two stages. First, the document is partitioned into a set of smaller XML files, which are then interlinked using XInclude [xin08] pointers. The optimal size of a partition is computed using a formula which considers the random versus sequential access characteristics of a hard disk. The second stage involves the parsing of a partitioned document. The key goal is to read a minimal set of partitions in order to perform the sequence of navigation commands. 2LP loads (pre-parses using the terminology of Figure 5.1) the partitions in a lazy manner, that is, only when they are absolutely necessary for the navigation sequence. In the case of DOM, we maintain an overall DOM tree $D(T)$ which is initially the DOM tree of the root partition P_0 of T. Then $D(T)$ is augmented with the DOM trees $D(P_i)$ of the loaded partitions P_i.

Further, to control memory usage, our approach also performs lazy unloading of inactive partitions (discussed in Section 5.3) if the total amount of main memory used by the DOM tree exceeds a threshold. Thus, in addition to a fast pre-parsing stage, our method also allows DOM-based parsing with limited memory resources. Note that previous lazy parsing techniques can also implement the proposed technique for optimizing memory usage, but to a smaller extent since the virtual document tree must be stored in memory at all times.

5.3 2LP on Partitioned XML Documents

Let T be the original XML document, and $P_0, \ldots P_n$ be the partitions to which T was split during the partitioning stage, explained in Section 5.4. P_0 is the root partition, since it contains the root element of T. Figure 5.3 shows an example of a partitioned XML tree. All the partitions are connected by XInclude elements, containing the Uniform Resource Identifier (URI) to the partition file. The XInclude elements are represented in the figure by nodes b', f' and j', as explained in Section 5.4.1.

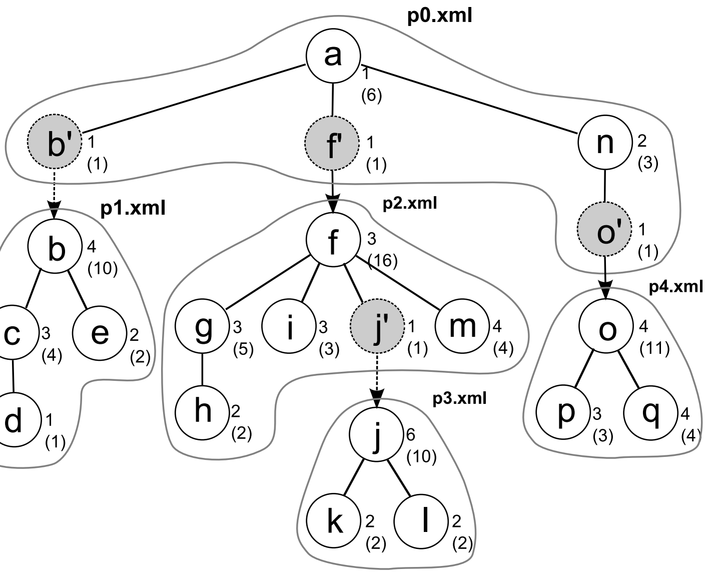


Figure 5.3: Partitioned XML Tree after partitioning the tree in Figure 5.2.

Note that by creating a partition (e.g., P_2), the key result is that we facilitate skipping the subtree rooted at this partition. That is, by creating partition P_2 we can directly access node n from node f'.

The XML representation of two of the partitions in Figure 5.3 is shown in Listing 5.1. Partition P_0 corresponds to the root partition since it contains the root of the original XML

Listing 5.1: XML Documents after partitioning.

```
<!-- p0.xml -->
<Catalog>
  <xi:include href="p1.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude" />
  <xi:include href="p2.xml"
    xmlns:xi="http://www.w3.org/2001/XInclude" />
  <Book title="XML Queries" year="2002">
    <xi:include href="p4.xml"
      xmlns:xi="http://www.w3.org/2001/XInclude" />
  </Book>
</Catalog>

<!-- p1.xml -->
<Book title="XML Databases" year="2002">
  <Chapter title="XML Introduction">
    <Section title="SGML" />
  </Chapter>
  <Chapter title="Semistructured Data" />
</Book>
```

document. The subtree rooted at the first `Book` element was partitioned and the `Book` element has been replaced by the XInclude pointer to the XML document of Partition P_1. This additional element added to the tree upon partitioning will hold the reference to the root of the partition's subtree. We explain this aspect in detail in Section 5.4.

Listing 5.2 describes the process of loading (pre-parsing) a partition. After loading a partition, progressive parsing occurs as needed. The `loadPartition()` method replaces, in the working DOM tree, the XInclude pointer element e with the DOM tree of the partition that e points to.

To ensure the double-lazy processing of the partitions, we need to decide when it is absolutely necessary for a partition to be loaded. Intuitively, a partition must be loaded when a navigation method (e.g., `getFirstChild()`) cannot be executed without doing so, that is, the return value of the method cannot be computed otherwise.

Listing 5.2: Load Partition algorithm.

```
procedure loadPartition(XIncludeElement $e$)
begin
  $newPartitionRoot \leftarrow$ preParse($e$.getAttribute("href"));
  replace($e$, $newPartitionRoot$); /* replace e by
                    newPartitionRoot in the node tree */
end
```

Similarly, we also need to decide which partitions to unload and when to do so in order to accommodate new partitions that need to be loaded, given that the available system memory is limited. We address unloading of partitions in detail in Section 5.5.

We now present the 2LP versions of the key DOM methods that may trigger the loading of a partition: `getFirstChild()`, `getTextContent()` and `getNodeName()`. Note that the `getNextSibling()` method cannot trigger a partition loading, because even if the sibling node is an XInclude pointer, we do not have to load the partition before the user asks for the details of the returned node (e.g., using `getNodeName()` shown below).

Figure 5.3 presents the `getFirstChild()` method with the logic to decide whether a partition has to be loaded. The original method only returns the `firstChild` member of the current object ("`this`"). In our modification, the loading is performed if the current node is an XInclude element, and it will assign the root element of the loaded partition to the firstChild member variable. Thus, instead of returning directly the first child of the XInclude node, we return the first child of the root element of the partition.

Example 3.1 *Consider the partitioned XML document depicted in Figure 5.3. Let's also consider the root-to-leaf navigation pattern $a \rightarrow f \rightarrow j \rightarrow k$. We start by parsing and traversing the root partition, labeled P_0. The first node-step, a, is satisfied in partition P_0, but to satisfy the second node-step, f, we need to follow the XInclude pointer to partition P_2, while completely skipping the processing of P_1. After pre-parsing partition P_2, we*

Listing 5.3: Modified `getFirstChild()` method to handle the lazy loading of partitions.

```
Node getFirstChild() {
  if this.isXIncludeElement() {
    loadPartition(this);
  }
  return firstChild;
}
```

progressively parse it to reach f. We need to satisfy the last two node-steps by following the pointer to partition P_3, pre-parsing it to then progressively parse the desired nodes. In this example, we omitted the traversal of partitions P_1 and P_4. □

***Example 3.2** Let's consider the XML document in Listing 5.1 and the XPath query* `/Catalog/Book[@title=''Storage Principles'']/Chapter`. *The careful reader can verify that this query requires loading all the partitions, even when we lazily process the document.* □

Note that in Example 3.2 we had to load partition P_1 just to read an attribute of its root element. To save such unnecessary partition loadings we extend the attributes of the XInclude element to contain additional information about the root element of the partition. This may save the loading of a partition when only information about its root node is required. Thus, the partition will be loaded only if the information needed by the navigation is not included in the pointer element. The data duplication to implement this idea is minimal, as shown in Section 5.7.2, since internal XML nodes typically are very small.

Table 5.4 summarizes the different *inclusion levels* based on the data from the partition's root element that is duplicated in the corresponding XInclude element. The names of the attributes used to store this data in the XInclude element are also displayed. For the TAG_ATR level, we use a single attribute whose value will resemble a query string (as used in World Wide Web forms) of the form $field1 = value1\&field2 = value2\&field3 = value3...$ [BL08].

Table 5.4: Attributes Stored for Different Inclusion Levels

Inclusion Level	Data to Include	Attribute Name
NONE	None	N/A
TAG	Tag *(Default)*	`xiPartitionTag`
TAG_ATR	Tag + Attributes	`xiPartitionAtr`
TAG_ATR_TXT	Tag + Attributes + Text	`xiPartitionTxt`

Example 3.2 (continued) *If we extend the XInclude elements depicted in Listing 5.1 according to Inclusion level TAG_ATR and execute the same XPath query, we will find the necessary information about the tag names and attribute values in the XInclude pointer elements. Thus, partitions P_1 and P_4 will not be processed at all, since the attribute values added to the XInclude pointer can help us discriminate which "Chapter" elements satisfy the attribute condition without loading the partition.* $\square$

In addition to the `getFirstChild()` method presented above, which is unaffected by the inclusion level, we now show how other key navigation methods of DOM need to be modified for the 2LP. Figure 5.4 presents two navigation routines that have been modified to allow the double-lazy processing of XML partitions with different inclusion levels. Similar to the `getFirstChild()` method, these two methods return (originally) just the corresponding member variable of the object. By modifying them, the methods will lazily include the corresponding partition if and only if this is needed to satisfy the navigation pattern and if the desired information is not included in the XInclude pointer element. If the inclusion is performed, the root element of the partition is assigned to current object ("`this`") and its member variables (name and text for `getNodeName()` and `getTextContent()` respectively) are returned. Similar modifications are performed for the other DOM methods that can potentially trigger the loading of a partition.

Listing 5.4: Key Modified Document Object Model Navigation Methods.

```
String getNodeName()
  if(this.isXIncludeElement()) {
    /* Check for tag information in XInclude element */
    if(inclusionLevel != NONE) {
      name = this.getAttribute("xiPartitionTag");
      /* The xiPartitionTag attribute inside the
         XInclude
       * element stores the tag name of the root element
       * of the partition */
    } else {
      /* Make ‘‘this’’ point at the root element of the
       * loaded partition, and update ‘‘name’’ variable
          */
      loadPartition(this);
    }
  }
  return name;
}

String getTextContent(){
  if(this.isXIncludeElement()){
    if(inclusionLevel == TAG_ATR_TXT) {
      text = this.getAttribute("xiPartitionTxt");
    } else {
      /* Make ‘‘this’’ point at the root element of the
       * loaded partition, and update ‘‘name’’ variable
          */
      loadPartition(this);
    }
  }
  return text;
}
```

5.4 Partitioning the XML File

Our main goal when partitioning XML documents is to minimize the 2LP parsing time needed for navigating the document.

In what follows, we first describe (in Section 5.4.1) how to partition an XML document by selecting subtrees of an optimal size. Then in Section 5.4.2 we make a theoretical analysis to obtain the optimal partition or subtree size, based in simplified navigation patterns.

5.4.1 Partitioning Algorithm

The key criterion to partition the original document is the number of blocks that each partition will span across the hard disk drive (i.e., the partition size). This size criterion is independent of the particular tree-structure (or schema if one exists) and the query patterns, and is shown to lead to efficient partitioning schemes (Section 5.7). The rationale behind this is that disk I/O performance is dictated by the average size of I/O requests when accesses are random [DR03].

The key idea of the algorithm is a bottom-up traversal of the XML tree, where nodes are added to a partition until the size threshold (in number of blocks) is reached. We show how the optimal partition size is calculated in Section 5.4.2.

Since we are using XInclude to simulate the physical pointers, we need to comply with the XInclude definition and hence provide partitions that are themselves well-formed XML documents. This means that our partitions need to have exactly one root element. Thus, the partitioning algorithm must include entire subtrees when creating a new partition. This constraint leads to having a few very large partitions since every XML document typically has very few nodes with very high fanout (e.g., open_auctions node in XMark [Fra04]). However, as we shall show in Section 5.7, this does not degrade the

Listing 5.5: Partitioning Algorithm.

```
int partitionTree(Node n, int threshold){
  /* Returns the size in bytes of the node n, including
     attribute names and values and text section */
  size = getSize(n);
  for(Node c : n.getChildren()) {
    size = size + partitionTree(c);
  }
  if(size >= threshold && !isRoot(n)) {
    createPartition(n);
    /* Recalculates the size after creating partition */
    size = getSize(n);
  }
  return size;
}

void createPartition(Node n) {
  x = createNewXMLFile();
  /* Replace subtree rooted at n in current XML document
     by an XInclude element pointing at file x */
  addXIncludePtr(n, x);
  /* Move the subtree rooted at n to file x */
  moveSubtree(n, x);
}
```

parsing performance since these partitions typically need to be completely navigated by XPath queries.

Figure 5.5 describes the basic tree partitioning algorithm. The `partitionTree( T.root, threshold )` method will recursively traverse T in a bottom-up fashion, calculate the size of each subtree, and if this size exceeds the threshold, then the `createPartition()` method is called for this subtree. The `createPartition()` method will move the entire subtree to a new XML document and a new XInclude element will replace its root node in the original XML file to reference the new partitioned subtree. Also, depending on the inclusion level flag, specific information of the partition's root element will be added to the newly created XInclude element.

Listing 5.1 shows the resulting partitioned XML tree for the XML tree of Figure 5.2 with a threshold of 10 blocks per partition. Node b' is the XInclude element which points to the partition rooted at node b. The same holds for nodes f', j', o'.

Example 4.1 *Consider the XML document in Figure 5.2. When we execute* `partitionTree( a, 10 )`*, the depth-first traversal of the tree rooted at a begins. The traversal will descend until it reaches the leftmost branch, and from there it will begin the bottom-up search for the subtree whose size in blocks is larger or equal to the specified threshold. Hence, we first create a new partition for the subtree rooted at node b, replacing this node with an XInclude pointer to the newly created partition. We assume in this case that we are using the default inclusion level (NONE), and thus an extra block is used by the pointer to maintain the data. We continue the navigation and create another partition with the subtree rooted at node j, repeating the same steps; we further create the new partitions rooted at nodes f and o.* □

5.4.2 Estimating the Optimal Partition Size

To obtain an appropriate value for the partition size, we conduct the following analysis for the root-to-leaf navigation pattern described in Section 5.2.1. In particular, we calculate the average access time to navigate from the root to each of the leaves of the XML document. While performing a similar analysis for general XPath patterns is infeasible due to the complexity and variety of the navigation patterns, we show, in Section 5.7, that using the theoretically obtained partition sizes leads to good results for general XPath queries as well.

We assume, for sake of simplicity, that our tree is complete and each node of T occupies a single disk block[2]. Therefore, the XML tree T, which has N nodes and degree d, has height $h = \log_d N$. As we shall see in the evaluation section, the simplifying assumptions used in our theoretical model do not significantly impact the key results; the theoretically optimal is found to be very close to the experimentally computed optimal size.

[2]Later on, we shall show that in spite of these simplifying assumptions, the experimentally obtained optimal partition sizes closely match our theoretical estimates.

Cost with no partitions: When the XML document is not partitioned (and hence 2LP is not applicable), the average cost of a root-to-leaf traversal is given by the following equation. Note that for simplicity we assume the document is parsed from scratch every time a navigation pattern occurs.

$$Cost^{noPart}_{root-leaf} = t_{rand} + N \cdot t_{transf} \tag{5.1}$$

where t_{rand} is the random access time needed to reach the root of the tree and t_{transf} is the time required to transfer one block of data for the specific disk drive. Note that the whole tree must be read (pre-parsed in Figure 5.1) to create the intermediate structure used to later progressively parse the document. No cost is assigned to the progressive parsing phase since the document has been already loaded in memory during pre-parsing.

Cost with partitions: Let us assume that the tree will be segmented into equally sized partitions, and we can describe each partition as having:

x: Number of nodes in partition

$h' = log_d x$: Height of the partition

In this case, the average cost for a root-to-leaf traversal is given by the following equation:

$$Cost^{Part}_{root-leaf} = (\,\#\, partitions\ accessed) \times (t_{rand} + x \cdot t_{transf})$$

where $t_{rand} x \cdot t_{transf}$ is the cost to pre-parse and load a partition. The number of partitions along a root-to-leaf traversal is h/h'. Hence we have the following equation:

$$Cost^{Part}_{root-leaf} = \tfrac{h}{h'}(t_{rand} + x \cdot t_{transf})$$

Observe that the ratio of heights can be simplified using logarithmic properties, and is independent of d. As a result, we obtain:

$$Cost^{Part}_{root-leaf} = \frac{\ln N}{\ln x}(t_{rand} + x \cdot t_{transf}) \tag{5.2}$$

Based on (5.2), we model the optimal cost of the partition size for four different hard disk drives, described in Table 5.5. A detailed description of our hard disk drive model and how we calculate the data transfer and random access times is included in Section 5.2.2.

Table 5.5: Disk Drive Characteristics

Disk Model	Size (GB)	t_{transf} (ms)	t_{rand} (ms)
Maxtor D740X	20.0	0.009446	5.441876
Fireball Plus	27.3	0.007688	5.359013
Cheetah 15K.4	36.7	0.002560	2.359615
Hitachi Ultrastar	73.4	0.003810	3.520530

Figure 5.4 presents the times $Cost^{Part}_{root-leaf}$ for the four different disk drives presented in Table 5.5 for varying partition sizes x. The optimal partition size is the value of x that minimizes the time.

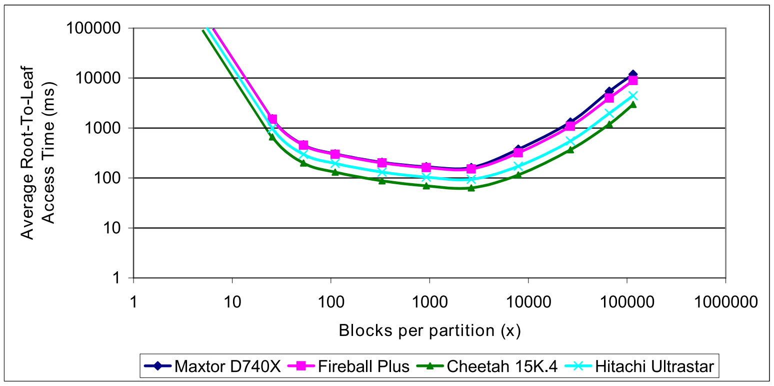


Figure 5.4: Effect of varying the partition sizes on the average root-to-leaf navigation access time.

The un-partitioned cost $Cost^{noPart}_{root-leaf}$ is equal to the time for the maximum partition size, where the whole document fits in a single partition.

5.5 Management of Limited Main Memory

As mentioned earlier, the DOM representation of an XML document can span up to five times its size in main memory. This fact combined to the increasing size of XML documents causes current XML parsers to often fail with an “Out of Memory” exception.

Listing 5.6: Modified Load partition Algorithm with Partition Unloading mechanism.

```
void loadPartition(XIncludeElement e) {
  newPartitionRoot = preParse(e.getAttribute("href"));
  /* replace e by newPartitionRoot in the DOM tree */
  replace(e, newPartitionRoot);
  registerPartition(this, e);
  while(size(T) > memory_threshold) {
    unloadPartition(getPartitionToUnload(T));
  }
}
```

We have created and implemented a mechanism to *unload inactive partitions* from the overall DOM tree. A partition P is considered as inactive if the path from the root of the DOM tree to the point of the current navigation sequence does not include any element from P.

To achieve the unloading of inactive partitions, we add a data structure that stores the information about the root element of the partition, the path of the partition document in the file system as well as a pointer to such root element. Every time a partition is loaded, all the metadata and the pointer are stored for further analysis. Also, after each partition is loaded, the system checks for the size of the overall DOM tree to decide whether one or more partitions have to be unloaded.

Listing 5.6 presents a modified version of the `loadPartition()` method presented in Listing 5.2. This new version adds the logic for unloading partitions to restrict the total amount of main memory used by the overall DOM tree to a fixed threshold. Every time a new partition is loaded, the method checks for the overall memory utilization and unload suitable partitions until the memory usage is below the threshold. Three auxiliary methods are added to handle the logic:

registerPartition(): This method receives as parameters the current element at which the partition has been added as well as the metadata of the partition. It stores the filename of the partition document in the file system and all the necessary information to recreate the XInclude pointer when the partition has to be unloaded.

getPartitionToUnload(): It analyzes the information stored by the `registerPartition()` method and decides which partition has to be unloaded. We implemented two variants of this method to implement the *First-in, First-out (FIFO)* and *Least Recently Used (LRU)* [SGG06] strategies, as used in the context of virtual memory page replacement. The *FIFO* strategy picks the oldest partition, taking the one at the head of the partition queue. When a partition is loaded, we insert it at the tail of the queue. Notice that the root partition, P_0, will never be unloaded, since it contains the root of the original XML document and we need to maintain that information accessible at all times. The *LRU* strategy discards the least recently used partitions; it requires keeping track of what was used when, which is more expensive than *FIFO*.

unloadPartition(): Once the `getPartitionToUnload()` method selects a partition, this method removes the underlying subtree from the overall DOM tree and reconstructs the XInclude pointer using the metadata stored by the `registerPartition()` method.

5.6 System Implementation

In this section, we describe the architecture and implementation of the two key components of our system: the XML Partitioner and the 2LP parser.

The system architecture is shown in Figure 5.5. The XML Partitioner takes a source XML document and partitions it based on a threshold determined using the model presented in the previous section. The 2LP Parser can also parse un-partitioned XML documents.

The 2LP parser was implemented by modifying the Xerces2 Java Parser, allowing it to handle the XInclude-defined partitions, but also preserving its backward compatibility. Figure 5.6 shows a simplified class diagram in Unified Modeling Language (UML) notation [BD03, Gro08], for the classes involved in our modification. The top layer is the W3C DOM Interface, followed by the Xerces2 Java Parser which is the implementation of such interface. The shadowed classes are the ones modified from the open-source package. The bottom layer is our own package, which encapsulates the modifications required to handle the partitioning and inclusion mechanisms.

Below, we describe the key ideas behind the modified and newly added classes in the implementation.

ElementImpl: This class was modified to handle inclusion behavior on the `getNodeName()` and `getAttributes()` methods. Depending on the inclusion level, these methods may answer

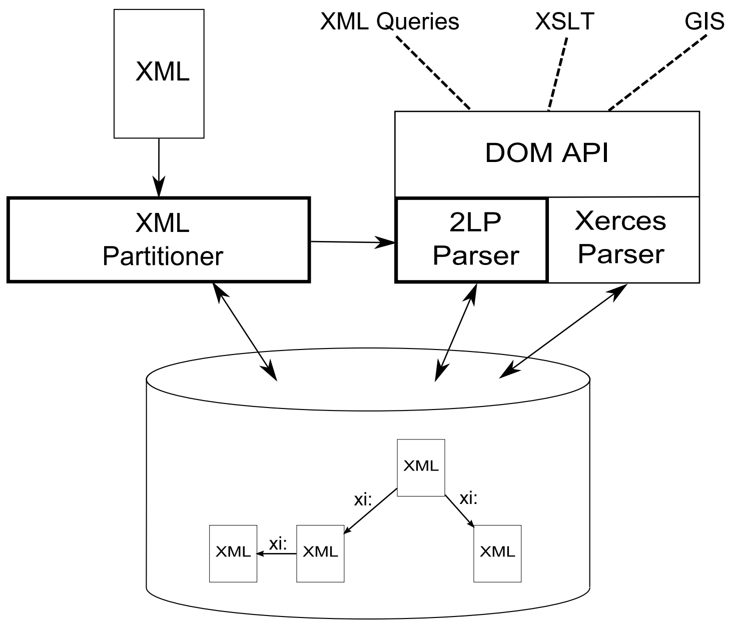


Figure 5.5: XML Partitioning and 2LP Architecture.

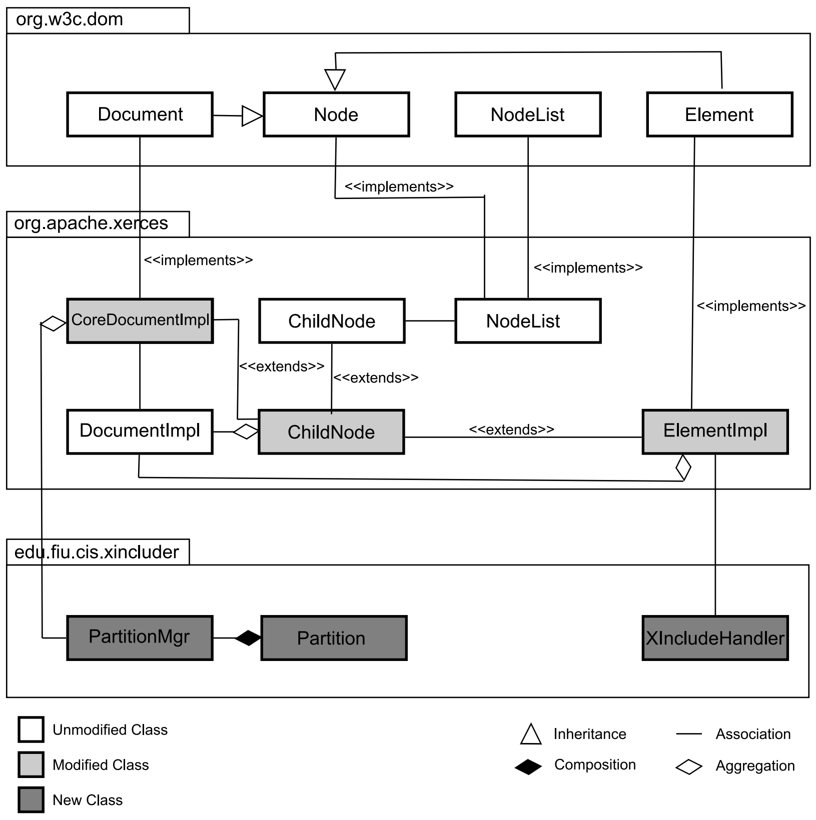


Figure 5.6: XML Partitioning and 2LP Class Diagram.

a query with local information or require an inclusion to import a new partition and answer the query.

PartitionMgr: The `PartitionMgr` class is attached to the `CoreDocumentImpl` class in the Xerces package, to manage the orchestration of traversal and inclusion. Every time a new partition is required, the `XIncludeHandler` will process the specified URI and a new `Partition` object will be created. It also manages the unloading mechanism.

XIncludeHandler: This class handles directly the inclusion operations when invoked from the `ParentNode` and `ElementImpl` objects in the Xerces package. This class works as a replacement to the default XInclude processor provided by the Xerces parser. In order to achieve this, we turn off the XInclude feature, and let our package handle these pointers.

Partition: This class is an abstraction to represent a partition processed by the `XIncludeHandler` class. Notice that all the user-level interaction is still performed via the DOM Interface, guaranteeing the backward compatibility desired as a design goal. We have made our XML Inclusion feature backward compatible, so another XML document that has XInclude pointers in it will be treated in the same way by our double lazy parser, and any partitioned document joined by XInclude pointers will be handled by any Xerces parser in a correct way.

5.7 Experiments

In this section, we evaluate our XML Partitioning and 2LP schemas. First, we experiment with optimal size of partitions based on the theoretical model proposed in Section 5.4.2. Second, we measure the performance of our techniques with two navigation patterns, root-to-leaf patterns and XPath queries, as presented in Section 5.2.1. Third, we evaluate the impact of our memory management optimization by unloading unnecessary partitions, as presented in Section 5.5.

Our framework was developed in Java using JDK 5.0. We modified the Xerces2 Java Parser 2.9.1 [xer08]. The experiments were performed on a 2.0GHz Pentium IV workstation with 512MB of memory running Linux. The workstation has a 20GB Maxtor D740X disk.

5.7.1 Evaluation of the Theoretical Model

We generated XML files of various sizes using the XMark generator [SWK$^+$02b]. We applied the partitioning algorithm to these documents, with several partition sizes (in blocks) to compare our theoretical model described in Section 5.4.2 against experimental results performing the same type of root-to-leaf navigation patterns described in Section 5.2. Note that throughout the experiments the 2LP parser is used for partitioned documents and the Xerces for un-partitioned.

Figure 5.7 shows the average time to traverse all the root-to-leaf paths for an XML document with XMark factor 0.5 (50MB), running on a Maxtor D740X hard drive as described in Section 5.4.2. The theoretical curves are based on the model presented in Section 5.4.2. Notice that the scale is logarithmic and the patterns of the graphs are similar, with a slight deviation in the experimental graph. We believe that the gap between the theoretical and experimental graphs is caused because the theoretical model does not take into account the processing time needed to navigate these paths and the effect of paging due to the limited amount of memory, but only the primary I/O time involved in reading the partition for the XML file. From the graph, we can infer the optimal size of the partition to be 2680 disk blocks, which is approximately one Megabyte.

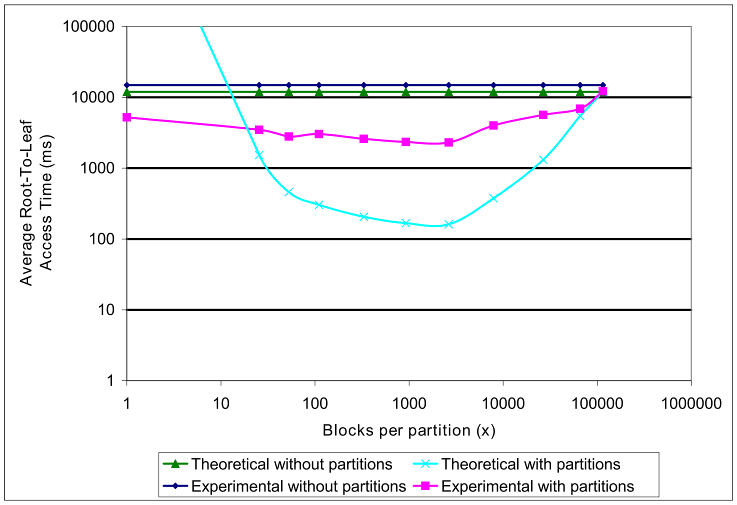


Figure 5.7: Average Traversal Time for Partition Sizes.

Next we compare the optimal partition size (obtained experimentally) for various document sizes (by varying the XMark factor) with the theoretical optimum. Figure 5.8 shows these results for the same hard drive, where again the theoretical and experimental values are close. For the first two XMark factors, the experimental optimal values are considerably smaller than the theoretical prediction. This is due to the fact that for the case of small files, having smaller partitions will benefit the performance of the navigation patterns, since it is more likely that the partitions (stored in the same directory) are contiguously placed on disk. The file system can efficiently (sequentially) retrieve all the partitions from the disk.

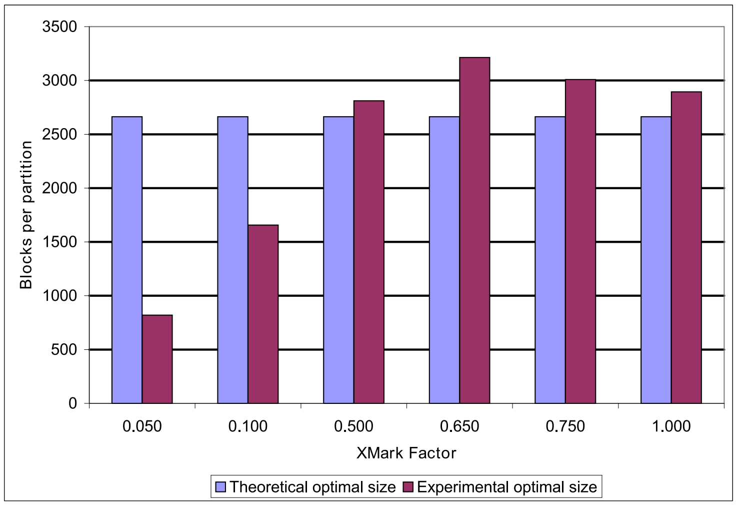


Figure 5.8: Optimal Size of Partitions.

5.7.2 Performance Evaluation

We now present the evaluation of our approach using two types of navigation patterns, root-to-leaf traversals and XPath queries. As explained in Section 5.4.2, the comparisons assume that the XML document has not been already parsed before a query or navigation pattern, that is, we measure both the pre-parsing and progressive parsing times of Figure 5.1. We measure three time components in the total execution time:

Pre-Parsing: The Xerces parser uses its deferred expansion node feature by initially creating only a simple data structure that represents the document's branching and layout. This phase requires scanning the whole document to retrieve this structure. For un-partitioned documents, it means that the first time we load the file, the whole document has to be traversed and processed; for partitioned documents, every time we process a new partition, it is pre-parsed to create the logical structure in memory.

Progressive Parsing: As the navigation advances, this initial layout built in the pre-parsing phase is refined, and all the information about the nodes is added to the skeleton. This phase is performed only on the visited nodes and will have the same behavior in both un-partitioned and partitioned documents.

Inclusion: This phase is introduced by the 2LP components, and captures the time required to include and import the new partition into the working document. This component does not apply to un-partitioned documents.

Root-to-leaf traversal cost: Figure 5.9 shows the average access cost in milliseconds for the root-to-leaf access patterns, comparing the performance for different XMark factors. To compute the average time, we sampled 10% of the leaves of each document, adding each tenth leaf into the sample, and performed. root-to-leaf traversals for each sampled leaf. A traversal in this case results in a sequence of parent-to-first-child and sibling-to-next-sibling operations in order to reach the desired leaf. These experiments were performed with the theoretical optimal partition size and the NONE inclusion level (the inclusion level does not impact the simple root-to-leaf traversals).

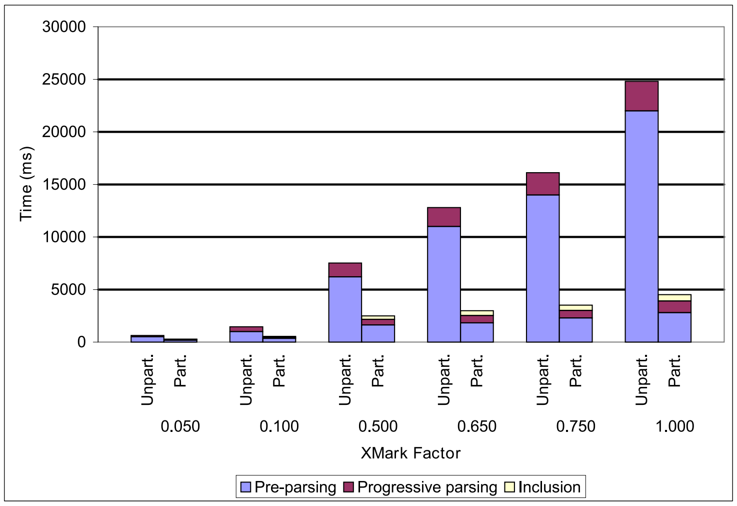


Figure 5.9: Root-To-Leaf Access Cost.

XPath query cost: Our second experiment executes a set of XPath queries over the XML data. The queries are shown in Table 5.6. We have included the performance queries from XPath-

Mark [Fra04], that is, the ones that test the execution time and not specific XPath functional aspects. We added more queries to have more reliable results.

Table 5.6: XPath Queries

#	Query
Q_1	$/site/closed\_auctions/closed\_auction/annotation/description/parlist/$ $listitem/text/keyword$
Q_2	$/site/people/person/watches$
Q_3	$/site/open\_auctions/open\_auction/annotation/description/text/keyword$
Q_4	$/site/people/person/address/country$
Q_5	$/site/regions/australia/item/description/tex/emph$
Q_6	$/site/people/person/*/business$
Q_7	$/site/closed\_auctions/closed\_auction/*/description$
Q_8	$/site/regions/*/item/description/text$
Q_9	$/site/open\_auctions/openauction$
Q_{10}	$/site/closed\_auctions$
Q_{11}	$/site/regions/australia$
Q_{12}	$/site/closed\_auctions/closedauction$
Q_{13}	$/site/regions/*/item$
Q_{14}	$/site/*/australia$
Q_{15}	$/site/open\_auctions/open\_auction[@id =' openauction0']/bidder$
Q_{16}	$/site/regions/asia/item[@id =' item4']/mailbox/mail/from$
Q_{17}	$//keyword$
Q_{18}	$/site/closed\_auctions//itemref$

For this set of experiments, we used several XML document sizes corresponding to various XMark factors. Once again, we use the theoretically optimal partition size for partitioning the XML documents. We used the default inclusion level (TAG) for these experiments.

Figures 5.10 and 5.11 show the performance of such queries for XMark factors of 0.5 and 1 (100MB) respectively. Figure 5.12 shows the average values for the same experiment over three datasets with XMark factors 0.500, 0.750 and 1.000. We see how for un-partitioned files, the pre-parsing time is always similar, since the whole document has to be processed to load the initial layout. For partitioned files, only the required partitions are processed, leading to significant reduction in the pre-parsing phase in most of the cases. We can observe that the partitioned documents perform consistently better than the un-partitioned ones. We have some cases in which the performance of the partitioned documents is almost equal to the performance of the original files. These cases, such as Q_3, Q_9, Q_{14} and Q_{15}, need to traverse most sections of the tree, requiring the inclusion of most partitions.

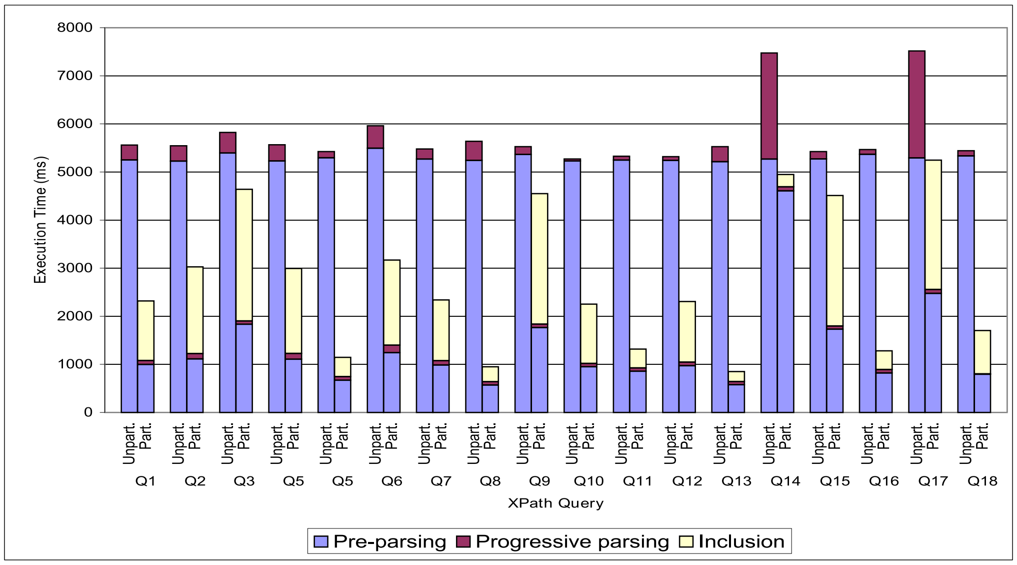


Figure 5.10: XPath Query Performance for XMark Factor = 0.5 using the performance XPath queries from XPathMark.

In the cases of Q_9, Q_{14} and Q_{17}, the open_auctions partition is loaded which has a size of 15MB (due to the fact that each partition must be a well-formed XML document, as explained in Section 5.4.1). Pre-parsing and progressively parsing this large partition penalizes these queries and they almost match the execution time of the un-partitioned version. However, in a typical scenario, such large partitions must be completely accessed anyways, except for the rare case when a navigation pattern specifies a child at a particular position (e.g., 1000^{th} child).

The inclusion time component varies correspondingly to the size of the partitions that have to be included into the working document. We see then that the inclusion component for Q_3, Q_9, Q_{14} and Q_{15} is large, but again this is caused by the large size of the open_auctions partition required to satisfy all these four queries. For these same queries we found large segments of time consumed by the Inclusion operation. The reason is that we rely on the `Document.importNode()` method provided by the DOM model which traverses the whole imported XML tree and updates

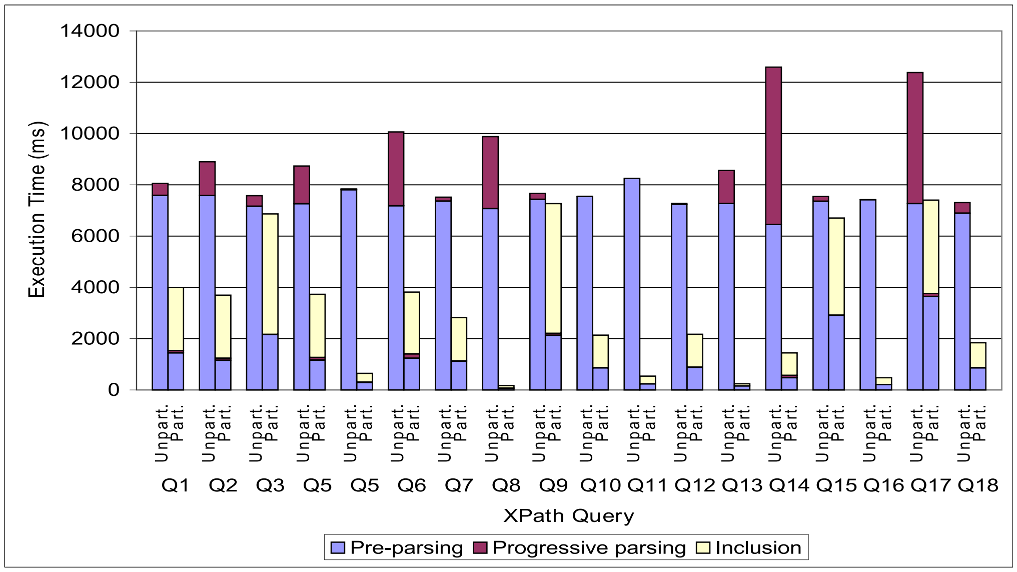


Figure 5.11: XPath Query Performance for XMark Factor = 1 using the performance XPath queries from XPathMark..

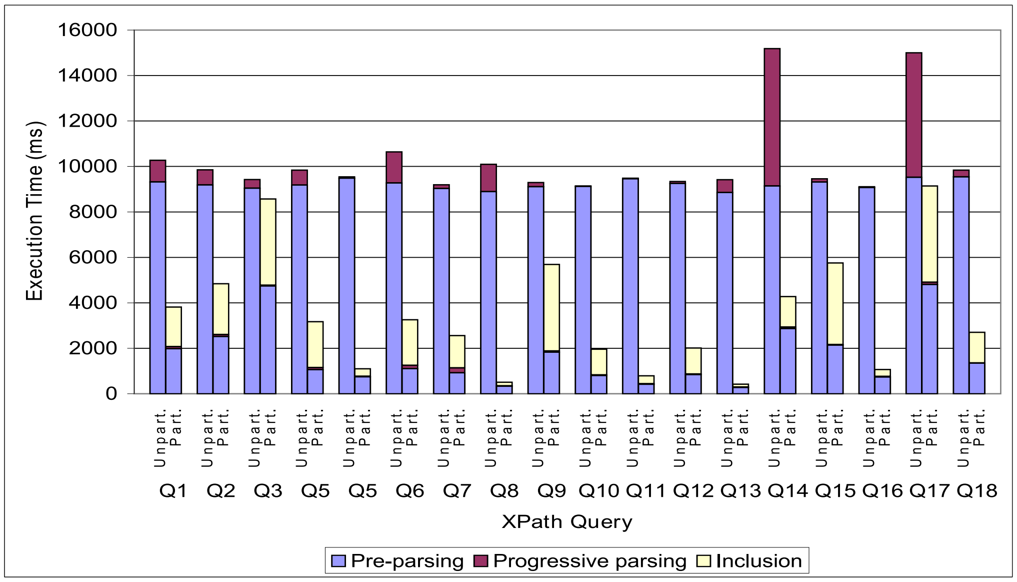


Figure 5.12: Average XPath Query Performance for XMark factors from 0.050 to 1.000, using the performance XPath queries from XPath-Mark.

the owner document for every single node. Even when the tree is already in memory, this operation is CPU intensive, delaying the process of including the new partition.

Inclusion Levels: We now experiment with the inclusion levels described in Section 5.3. Initially, we observe the increase of space required by the partitions given the distinct inclusion levels, compared against the original unpartitioned document. We partitioned a document with XMark Factor = 0.5.

Figure 5.13 presents the results of this experiment, showing low space overhead even when the full information of the partition root is added to the XInclude element. Compared to the size of the original size and to the size of the partitioned file with inclusion level NONE, we can say that practically no overhead exists. We can see how the third inclusion level has the same overhead as the second one. This is due to the fact that most of the nodes that contain text are leaf nodes, and none of the internal nodes that were chosen to root a new partition contain text values.

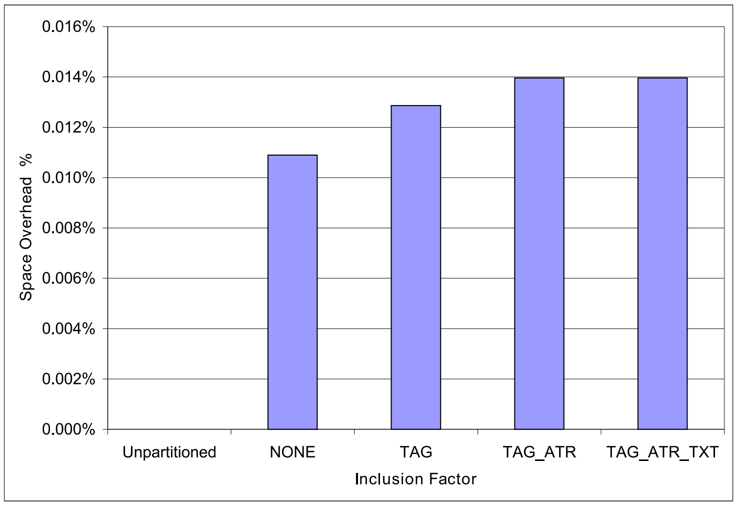


Figure 5.13: Space Overhead for Inclusion Levels.

Figure 5.14 shows the average query execution time performance when XPath queries are executed over partitioned documents with different Inclusion levels. We picked several XPath queries that represent different categories of queries and different axes. Given the practically inexistent space overhead discussed above, adding information about the root element of the partition in the XInclude physical pointer can give us a significant percentage of gain. In particular, the TAG_ATR level is generally the best choice.

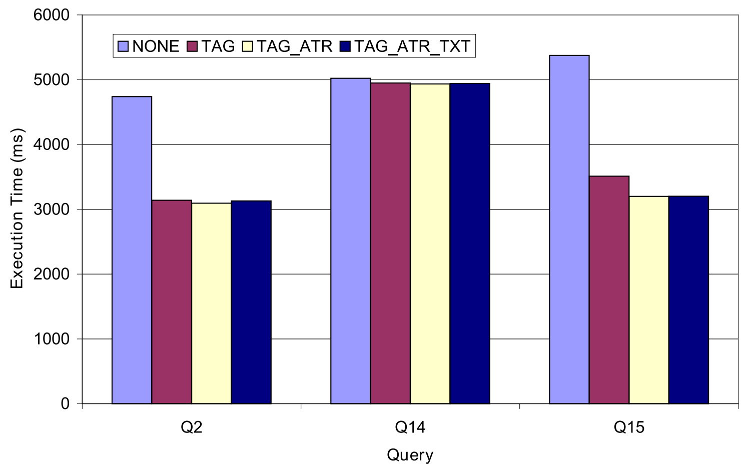


Figure 5.14: Execution Time with Inclusion Levels.

5.7.3 Partition Unloading

In this section we evaluate the performance of 2LP when the amount of main memory designated to store the DOM tree is limited and hence, the partition unloading mechanism is required. We simulated this by introducing a *total memory threshold factor* M_T, which takes into account the DOM overhead claimed by [NJ03] which concludes that a DOM document can expand in main memory up to three or five times the size of the XML file. This factor models the limited number of Megabytes that can be allocated by the DOM document at any given moment.

For this experiment, we repeated the execution of our performance evaluation XPath queries, but this time adding the Partition Unloading mechanism to our 2LP. The XPath queries from Table 3 were executed sequentially, without resetting the DOM tree to the initial partition P_0, this with the objective of having several partitions loaded before each query was executed. To simulate the *Total Memory Threshold* M_T, we set the Java Virtual Machine's maximum java heap size to 450MB, and used our partitioned XML document for xmark factor = 1.0.

Figure 5.15 shows the total amount of main memory allocated by 2LP after each query is executed. The JVM Memory Limit resembles the *total memory threshold factor* M_T, as limited by the JVM maximum heap size. We measured the performance of 2LP without Unloading mechanism as well as the behavior of the unloading mechanism following two strategies: *First-In-First-Out (FIFO)* and *Least-Recently-Used (LRU)*, both as used in the context of main memory page replacement. Both strategies restrict the loaded partitions according to M_T, replacing the appropriate partition as dictated by each strategy.

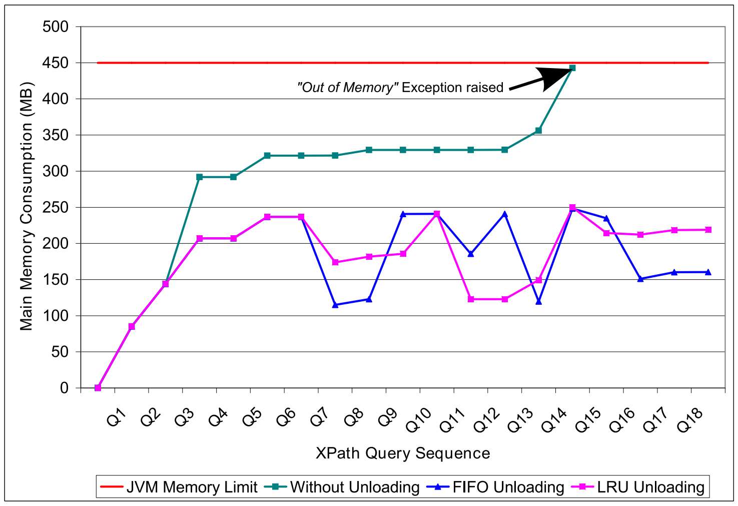


Figure 5.15: Loaded Partitions after Sequential Query Executions.

The execution of 2LP without Unloading followed a behavior as shown in Figure 5.11, where at a point after the execution of Q_{14}, the application crashed with an “Out of Memory” exception, not being able to perform the total execution of our query load. Using a lazy parser like Xerces on the unpartitioned document leads to the same behavior. This was due to the fact that Q_{14} uses a wildcard and requires almost the whole tree to be loaded into main memory. In contrast, both *FIFO* and *LRU* approaches for the unloading strategy were able to manage the critical point of loading several partitions during query Q_{14}, working properly until the last query was executed even under the main memory limitations. We can see that our unloading approach has the potential to scale better to parse large documents under limited memory conditions, whereas current approaches including Xerces will raise “Out of Memory” exceptions.

Both *FIFO* and *LRU* strategies lead to similar behaviors, with slight differences in the order in which the partitions are unloaded as shown for queries Q_7 and Q_{12}; in Q_7 a larger partition is unloaded by the *FIFO* strategy, whereas *LRU* unloads a smaller one. Similarly, for Q_{12}, the *LRU* strategy selects a large partition to unload, while the partitions unloaded by *FIFO* are not as large.

Figure 5.16 compares the execution time of the XPath queries when the 2LP utilizes the Unloading Mechanism. The figure contains the execution times for 2LP with no Unloading Mechanism, as well as both *FIFO* and *LRU* strategies. The execution of the queries was performed sequentially as explained before, and this caused the first two queries to perform similarly under the three conditions, since the same partitions have to be loaded in the same order to solve the query. For the execution of query Q_3, the size of the overall DOM tree has surpassed the memory threshold and hence one partition has to be unloaded, meaning a penalty in the total execution time. Queries Q_4 and Q_6 show a similar performance for the three variants, since all the partitions that are needed to solve these queries are already loaded into memory in these specific moments, not needing to parse any new partitions. Also none of the currently loaded partitions were unloaded by these queries. In the case of queries Q_{14} and Q_{17}, the maximum memory threshold was reached several times during the query execution, given the large number of partitions required to be parsed by the 2LP. This causes a lot of partitions to be unloaded during the query execution, drastically penalizing the total execution time. Queries Q_3 and Q_9 need to navigate the open_auctions subtree, requiring a larger amount of processing time given the large size of such subtree.

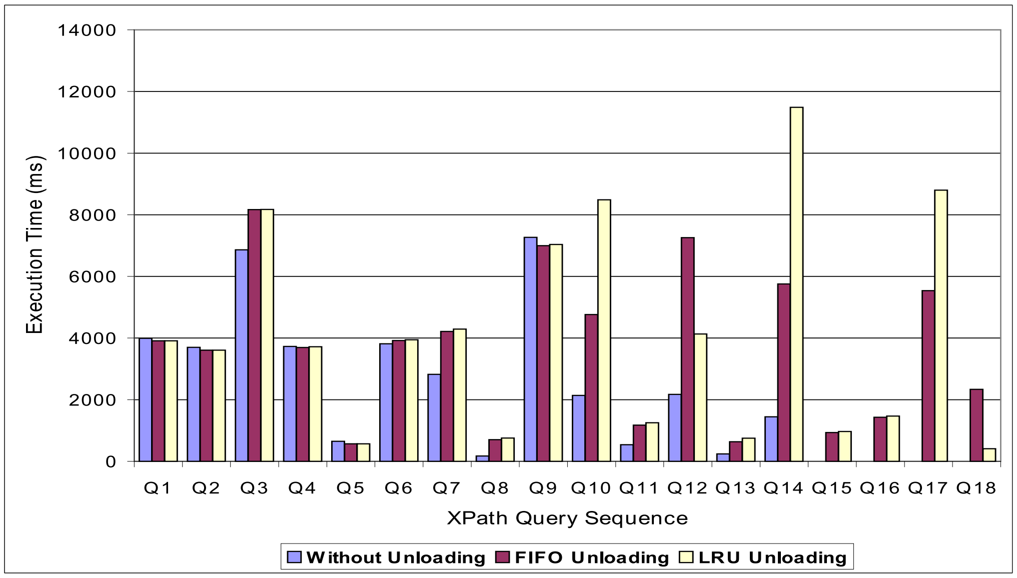


Figure 5.16: Execution Time with Unloading Mechanism.

Figure 5.17 shows the number of partitions that are loaded, re-loaded and unloaded during the execution of each XPath query. A loaded partition means that it has been parsed for the first time by the 2LP. An unloaded partition is one that has been chosen by the Unloading Mechanism to be discarded. A re-loaded partition is one that has been previously discarded but it is needed to satisfy the query and hence is parsed again.

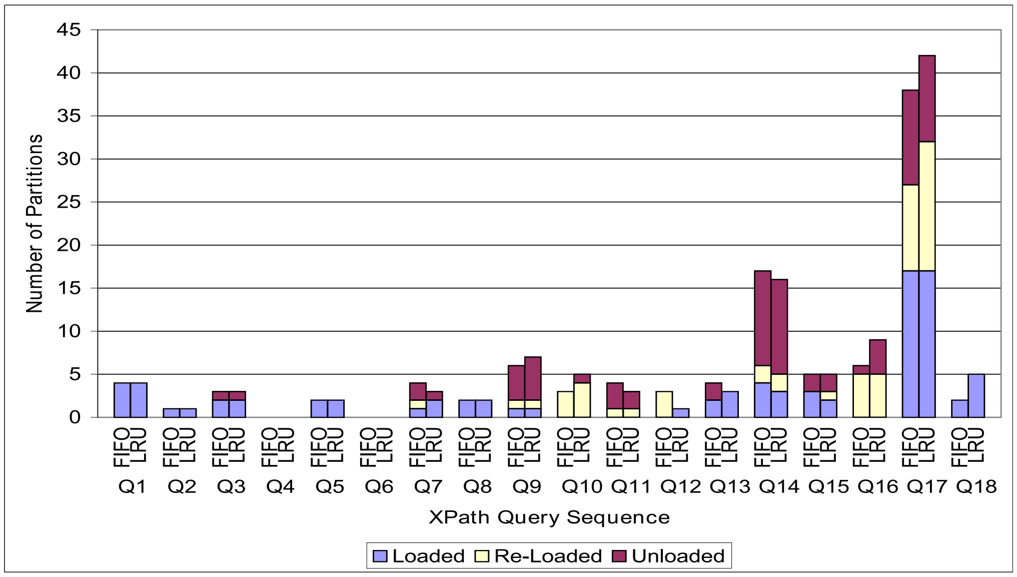


Figure 5.17: Loaded, Re-Loaded and Unloaded Partitions.

Again we can see how the performance of queries Q_1, Q_2, Q_3 and Q_4 is similar; the same partitions have to be loaded and unloaded for these cases. As observed in Figure 5.16, the execution of queries Q_4 and Q_6 do not require the parsing of any new partitions, since all the necessary partitions are already in main memory. The performance of queries Q_{14} and Q_{17} is also related to the behavior in the previous figure. The wild cards and descendant operators require a large number of partitions to be parsed and with this, a large number of partitions to be unloaded as well.

We can also see that both the *FIFO* and *LRU* strategies behave similarly in terms of re-loaded partitions. In terms of total execution time, *LRU* is penalized by the reordering of the partitions in the internal data structures of the strategy.

5.8 Related Work on Parsing of Semistructured Documents

Nicola and John [NJ03] have identified the XML parsing process as a bottleneck to enterprise applications. Their study compares XML parsing in several application domains to similar applications that use relational databases as their back-end. Operations such as shredding XML documents into relational entities, XPath expression evaluation and XSLT [xsl08, XSL07] processing are often determined by the performance of the underlying XML parser [NJ03], limiting the massive adoption of native XML databases into large-scale enterprise applications.

Noga, Schott, and Löwe [NSL02] present the idea of Lazy Parsing as presented in the Section 5.1. The virtual document tree can potentially be stored on disk to avoid the pre-parsing stage; however, the virtual document tree has to still be read from disk. Schott and Noga apply these ideas to the XSL transformations [SN03]. Kenji and Hiroyuki [KH05] have also proposed a lazy XML parsing technique applied to XSLT stylesheets, constructing a pruned XML tree by statically identifying the nodes that will be referred during the transformation process.

Lu et al. [LCP06] present a parallel approach to XML parsing, which initially pre-parses the document to extract the structure of the XML tree to then perform a parallel full parse. This parallel parsing is achieved by assigning the parsing of each segment of the document to a different thread that can exploit the multi-core capabilities of contemporary CPU's. Their pre-parsing phase is more relaxed than the one proposed by [NSL02] and that we use throughout our work; this relaxed pre-parsing only extracts the tree shape without additional information, and is used to decide where to partition the tree to assign the parsing sub-tasks to the threads. This partitioning scheme differs from ours since it is performed after the pre-parsing phase is executed, whereas ours is performed a priori, with the objective of optimizing such pre-parsing stage.

There have been efforts in developing XML pull parsers [xpu08] for both SAX and DOM interfaces. Also, [xpp08] presents a new API built just one level on top of the XML tokenizer, hence claiming to be the simplest, quickest, and most efficient engine for processing XML.

Huang et al. [HCL05, HCLL06] present a pre-filtering framework to improve the efficiency of XPath processing over large XML documents with the existing DOM and SAX models. Their framework utilizes an inverted index and a tiny search engine that locates the useful fragments that may be candidates to satisfy the input XPath query, and only these fragments are submitted to the XML parser. In contrast to our approach which is minimally invasive and is compatible with current XML parsers and standards, they use specialized proprietary storage and processing mechanisms.

Van Lunteren et al. [vLEB$^+$04] propose a programmable state machine technique that provides high performance in combination with low storage requirements and fast incremental updates. A related technique has been proposed by Green, Miklau, Onizuka and Suciu [GMOS02], to lazily convert an XPath query into a Deterministic Finite Automata (DFA). After this conversion is performed, they submit the XML document to the DFA in order to solve the query. They propose a lazy construction opposed to an eager creation, since constructing the DFA with the latter technique can lead to an exponential growth in the size of the DFA.

Kiselyov [Kis01] presents techniques to use functional programming to construct better XML Parsers.

Kanne and Moerkotte [KM06] have worked on tree partitioning algorithms, but their techniques are more oriented to low-level disk placement, mapping each partition to a single block on the disk drive to be further exploited by native XML data stores like Natix [Nat06].

Several works have been proposed in the area of XML compression. Some of these works [FLMM06, LS00] require the document to be decompressed before any query or navigation can be performed over the XML data. Some others, considered *query-friendly* [BLM05], only require a small subset of the document to be de-compressed. Some recent works [TH02, WLS07, DRR08] can support navigation in the compressed document. SDOM [DRR08] proposes a succinct way of representing XML documents in order to reduce their memory fingerprint and allow efficient navigation. However, SDOM still incurs the pre-parsing cost. Furthermore, their representation is not backwards compatible with current XML parsers. [LS00, TH02, WLS07, BLM05, FLMM06] have similar limitations. These XML compression and parsing techniques could be viewed as complementary to our work since we mainly optimize the pre-parsing stage with a slight optimization of the progressive parsing stage and they mainly optimize the latter one.

5.9 Conclusions

Lazy XML parsing is a significant improvement to the performance of XML parsing but to achieve higher levels of performance there is a need to optimize the pre-parsing phase during which the whole document is read. In this chapter, we address this problem by enabling laziness in the pre-parsing phase as well. To do so, we have proposed a mechanism to add physical pointers in an XML document by partitioning the original document and linking the partitions with XInclude pointers. We have also proposed 2LP, an efficient parsing algorithm for such documents, that implements pre-parsing laziness. Additionally, we implemented a dynamic partition unloading mechanism that can enables parsing in memory-limited systems, allowing us to parse and navigate large documents under conditions wherein other parsers typically fail. To aid partitioning decisions, we have proposed a theoretical model for the processing of partitioned documents and presented methods to compute optimal partition sizes. We have experimentally showed that 2LP outperforms other deferred evaluation techniques such as Xerces Java Parser.

CHAPTER 6

CHALLENGES FOR INFORMATION DISCOVERY ON ELECTRONIC HEALTH RECORDS

6.1 Motivation

The National Health Information Network (NHIN) and its data-sharing building blocks, RHIOs (Regional Health Information Organizations), are encouraging the widespread adoption of electronic medical records for all hospitals within the next five years. In addition, The Department of Health and Human Services (HHS) has recently increased funding and placed pressure on the healthcare industry to improve the technology involving the exchange of medical information. Many standards and protocols have been introduced that will aid in the process of unifying the electronic medical record into a single architecture. A key component of this effort is the adoption and standardization of *Electronic Medical Records (EMR)*. To date, there has been little or no effort to define methods or approaches to rapidly search such documents and return meaningful results.

One of the most promising standards for EMR manipulation and exchange is Health Level 7's Clinical Document Architecture (CDA) [CDA07], which leverages a semi-structured format (Extensible Markup Language, or XML), dictionaries, and ontologies to specify the structure and semantics of EMRs for the purpose of Electronic Data Interchange (EDI). This HL-7 architecture has been adopted worldwide.

The definition and adoption of this standard presents new challenges to related computer science disciplines like data management, data mining and information retrieval. In this chapter we study the problem of facilitating *information discovery* on a corpus of CDA documents, i.e., given a question (query) and a set of CDA EMRs, find the entities (typically subtrees) that are "good" for the query, and rank them according to their "goodness" with respect to the query. The success of Web search engines has shown that keyword queries are a useful and intuitive information discovery approach. Therefore, we focus in keyword queries in this chapter. Other types of information discovery queries on EMRs not studied here include numeric conditions, aggregation and statistics, classification and clustering (the last two are closer to the data mining discipline).

As an example, consider the usual scenario where a doctor wants to check possible conflicts or complications between two drugs. Keyword query “drug-A drug-B death” could be submitted to discover cases where a patient who took both drugs died. Note that the word “death” can be specified in many different elements of a CDA document, and also synonyms or related terms like “mortality” can be used instead. The latter can be tackled by leveraging appropriate medical ontologies like SNOMED Clinical Terminology (SNOMED CT) [SNO08] as discussed below.

To study the challenges and requirements of information discovery on EMRs we have built a diverse research team consisting of computer scientists, medical research doctors and a partner from the medical informatics industry. The medical doctors provided the domain knowledge regarding the types of queries and answers that are of interest as well as the possible applications of such an information discovery system. Furthermore, they enumerated the different critical dimensions in searching EMRs, like time, location, and type of stakeholder. These dimensions have not been considered in systems for searching general XML documents; however, ignoring these dimensions would significantly limit the use of an EMR information discovery engine.

The key ranking criteria found in current systems as well as the bibliography [Sal89, BYRN99, GSBS03] are (a) relevance, (b) quality (authority) and (c) specificity. Relevance to the query has the obvious meaning, while quality represents the query-independent importance of a result. For example, a medication is more important than the name of an insurance company for a clinical researcher. Specificity determines how focused a result is to the query. For example, returning a department of a hospital when the query is only relevant to a particular doctor of this department is worse than returning this doctor object.

It is challenging to define the information discovery semantics for CDA documents such that the three aforementioned key ranking criteria are considered, given the hierarchical structure and specific semantics of CDA, and the common references to outside entities like dictionaries, ontologies, separate text, or multimedia patient data. Medical dictionaries and ontologies typically used in CDA are SNOMED CT [SNO08], Logical Observation Identifiers Links and Codes

(LOINC) [Log06] and RxNorm [RxN07]. We also study how previous work on information discovery on XML data [AYBS04, AYCD06, CKKS05, CMM$^+$03, FG01, GSBS03, HGP03, HP06, LYJ04, XQu07, XP05] can be leveraged, and what limitations might exist in this unique domain.

We note that our study does not address the important *privacy issues* involved in accessing patient information, as required by the United States Health Insurance Portability and Accountability Act (HIPAA) [Hea08]. We envision two possible scenarios. The simplest scenario is that each division of an institution deploys the information discovery engine on its own corpus of EMRs and provides authentication-controlled access to the division's practitioners. The more complex scenario, which is out of the scope of this study, is to provide information discovery on a set of interconnected federated databases where elaborate access control mechanisms must be employed [BGBJ05].

The rest of this chapter is organized as follows: Section 6.2 presents a background exposition of current clinical information standards and a brief survey on information discovery on XML data. Section 6.3 addresses the challenges that we have identified to execute information discovery on a corpus of EMR documents. Section 6.4 presents additional related work. Our concluding remarks are presented in Section 6.5.

6.2 Background

In this section we review key standards used to represent clinical data and EMRs and present previous work on information discovery on general XML documents. In particular, Section 6.2.1 introduces some popular clinical information representation standards as well as clinical ontologies, whereas Section 6.2.2 presents the Clinical Document Architecture (CDA), which will be the focus of this chapter. Given that CDA is represented in XML, Section 6.2.3 presents a brief survey on information discovery on general XML documents.

6.2.1 Clinical Information Models, Dictionaries and Ontologies

The work in [Her02, KOG$^+$01] described Medical Informatics as the broad term representing the core theories, concepts and techniques of Information applications in health. We describe the key standards, dictionaries and ontologies that are currently used in CDA. In particular, we first present the *Reference Information Model (RIM)* [HL708b], the model from which the CDA documents derive their meaning. Three popular clinical dictionaries/ontologies referred to in CDA documents are presented - the *Systematized Nomenclature of Human and Veterinary Medicine (SNOMED)* [SNO08], the *Logical Observation Identifiers Names and Codes (LOINC)* [Log06] and *RxNorm* [RxN07].

Health Level Seven (HL7): Health Level Seven (HL7) [HL708a] is a not-for-profit organization that provides standards for interoperability in the healthcare industry, mainly focused on clinical and administrative data. HL7 is an American National Standards Institute (ANSI) -accredited Standards Developing Organization (SDO) that includes providers, vendors, payers, consultants, government groups and other entities interested in developing clinical and administrative standards for healthcare.

HL7 standards specify a series of flexible standards to facilitate the communication between heterogeneous systems and vendors, allowing information to be shared and processed in a uniform and consistent manner. During the years, HL7 has developed Conceptual Standards (i.e. HL7 RIM), Document Standards (i.e. HL7 CDA), Application Standards (i.e. HL7 CCOW) and Messaging Standards (i.e. HL7 v2.x and v3.0). These standards define the language, structure and data types that participate in the integration of heterogeneous systems [Cal08].

Reference Information Model (RIM): The HL7 Reference Information Model (RIM) is the grammatical specification of HL7 messages, constituting the building blocks of the language entities and the relationships among them. RIM can be represented as a network of classes, expressed using a notation similar to the Unified Modeling Language (UML) [Uni07]. Its structure can be summarized into six "core" classes and a set of relations between them, as depicted in Figure 6.1. We include a brief description of each class as follows:

The *Act* class represents all the actions and happenings –analogous to a verb– to be documented through the healthcare process, capturing all the events that have happened in the past, that are currently happening or that are expected to happen in the future. The terms ‘Act’, ‘Action’, and ‘Activity’ are all used interchangeably [HL708b].

The *Entity* class represents any physical thing or being –analogous to nouns– that takes part or is of interest in the health care and that is capable of participating in an Act. Although it instantiates any physical thing or group of physical things (including living subjects and organisms), it does not include the roles that things can play or the acts that things can perform.

The *Role* class ties an entity to the acts that it plays or provides, specifying how a particular entity participates in a particular act. Each role is played by one entity, but one entity in a particular role can participate in an act in several ways.

The *RoleLink* class specifies the connections and dependencies that exist between two different and individual Role objects. The Participation class specifies a relationship between a particular Role instance and a particular Act instance. At the same time, it connects the Entity playing the Role, to the specified Act, thus expressing the context for the Act in terms of who performed it.

The *ActRelationship* class associates a pair of Act objects,representing a connection from one Act to another one. Such relationships include “Act to Act” associations, as well as “Source/Target” associations between the objects. [HL708b] states that “ActRelationship on the same source Act are called the “outbound” act relationships of that Act. ActRelationships on the same target Act are called the “inbound” relationships of that “Act”. Table 6.1 presents some examples to each core class of the RIM model.

Each Act may be related to any number of Participations, in Roles, played by Entities, at the same time that each Act may be related to other Acts via the ActRelationship class. The Act, Role and Entity classes may also be specialized into other classes. As an example, the Entity class specializes into the class Living Subject, which itself has a specialization class called Person. Person then inherits the attributes of both Entity and Living Subject. CDA documents (Section 6.2.2) use the semantic definitions from the HL7 RIM, using the HL7 Version 3 Data Types [HL707b] to define the clinical content of the documents.

Since HL7 mainly focuses on information interchange, RIM also provides a set of classes to

Table 6.1: RIM Core Classes Examples

Core Class	Example
Act	Clinical observation Assessment of health condition
Entity	Person Chemical substance
Role	Patient Employee
RoleLink	Manager has authority over Analyst (Using role link for "direct authority").
Participation	Surgeon Author
ActRelationship	Theophylline mitigates asthma (Using ActRelationship of type "mitigates").

define a communication infrastructure, including Message Control and Infrastructure (structured documents and components) [HL708b, HL707a].

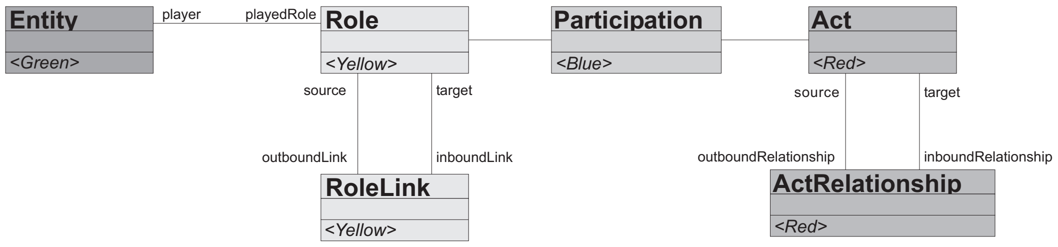


Figure 6.1: RIM Core Class Diagram.

Systematized Nomenclature of Medicine (SNOMED): The International Systematized Nomenclature of Human and Veterinary Medicine (SNOMED) was created more than 20 years ago as the conjunction of SNOMED RT and the United Kingdom's Clinical Terms Version 3, and has grown up into a comprehensive set of over 150,000 records in twelve different chapters or axes. These concepts are organized into anatomy (topology), morphology (pathologic structure), normal and abnormal functions, symptoms and signs of disease, chemicals, drugs, enzymes and other body proteins, living organisms, physical agents, spatial relationships, occupations, social contexts, diseases/diagnoses and procedures [SNO08]. Within the disease/diagnosis axis, many disease concepts have cross-references to other concepts in the terminology that are essential characteristics of the disease. These form a useful basis for further formalization and development of a reference terminology [Spa97].

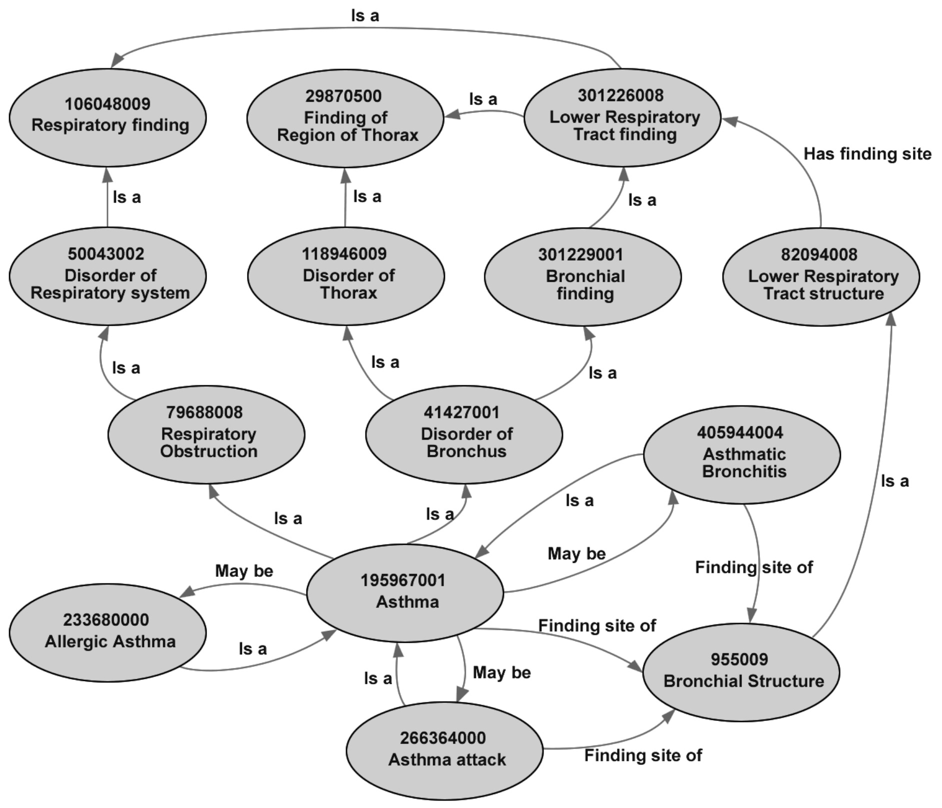

Is a
106048009 Respiratory finding
29870500 Finding of Region of Thorax
Is a
301226008 Lower Respiratory Tract finding
Has finding site
Is a
Is a
Is a
50043002 Disorder of Respiratory system
118946009 Disorder of Thorax
301229001 Bronchial finding
82094008 Lower Respiratory Tract structure
Is a
Is a
Is a
79688008 Respiratory Obstruction
41427001 Disorder of Bronchus
405944004 Asthmatic Bronchitis
Is a
Is a
Is a
Is a
May be
Finding site of
May be
195967001 Asthma
233680000 Allergic Asthma
Is a
Finding site of
955009 Bronchial Structure
Is a
May be
266364000 Asthma attack
Finding site of

Figure 6.2: Partial SNOMED ontology for the term “Asthma”

SNOMED has created and is committed to spreading the adoption and implementation of *SNOMED Clinical Terms (SNOMED CT)*. SNOMED CT is a universal health care terminology and infrastructure, whose objective is making health care knowledge usable wherever and whenever it is needed. It provides a common language that enables a consistent way of capturing, sharing and aggregating health data across specialties and sites of care. The SNOMED CT structure is concept-based; each concept represents a unit of meaning, having one or more human language terms that can be used to describe the concept. Every concept has inter-relationships with other concepts that provide logical computer readable definitions, including hierarchical relationships and clinical attributes. Figures 6.2 and 6.3 show sub graphs of the SNOMED CT ontology graph.

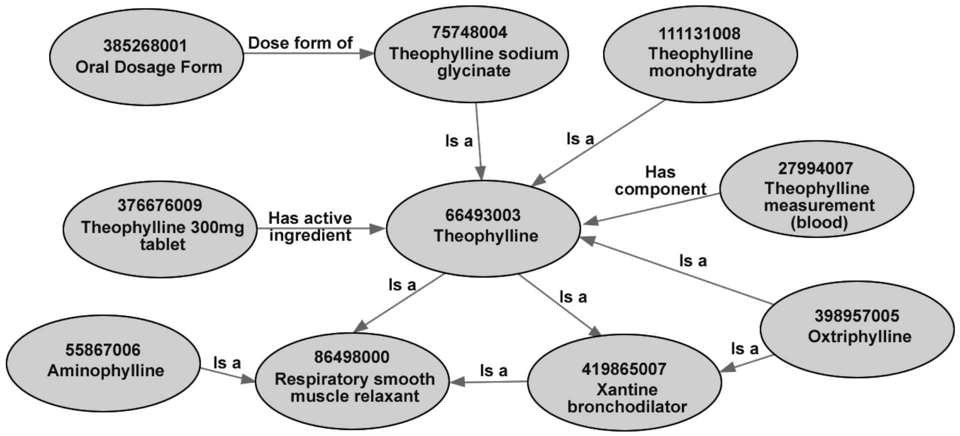


Figure 6.3: Partial SNOMED ontology for the term “Theophylline”

At the moment, SNOMED CT contains more than 325,000 concepts, with 800,000 terms in English, 350,000 in Spanish and 150,000 in German. Also, there are 1,200,000 relationships connecting these terms and concepts.
SNOMED CT terms are routinely referenced in CDA documents by their numeric codes, that is, the SNOMED CT vocabulary is referenced as an external domain according to HL7 V3 processes.

Logical Observation Identifiers Names and Codes (LOINC): LOINC is a voluntary effort housed in the Regenstrief Institute, associated with Indiana University. It was initiated in 1994 by the Regenstrief Institute and developed by Regenstrief and the LOINC committee as a response to the demand for electronic movement of clinical data. LOINC facilitates the exchange and pooling of results, such as blood hemoglobin, serum potassium, or vital signs, for clinical care, outcomes management, and research. Currently, most laboratories and other diagnostic services use HL7 to send their results electronically from their reporting systems to their care systems. However, most laboratories and other diagnostic care services identify tests in these messages by means of their internal and idiosyncratic code values. Thus, the care system cannot fully “understand” and properly file the results they receive unless they either adopt the producer’s laboratory codes (which is impossible if they receive results from multiple sources), or invest in the work to map each result producer’s code system to their internal code system. LOINC codes are universal identifiers for laboratory and other clinical observations that solve this problem.

The LOINC laboratory terms set provides a standard set of universal names and codes for identifying individual laboratory and clinical results. LOINC codes allow users to merge clinical results from many sources into one database for patient care, clinical research, or management. The LOINC database currently contains about 41,000 terms, which include 31,000 observational terms related to laboratory testing.

Each record in the LOINC database identifies a clinical observation and contains a formal 6-part name, a unique name for tests, identifying code with check digits, synonyms, and other useful information.

Currently, LOINC codes are being used in the United States by laboratories and federal agencies and are part of the Health Insurance Portability and Accountability Act [Hea08] Attachment Proposal [MHS+03]. Internationally, LOINC has been adopted in Switzerland, Hong Kong, Australia, Canada and Germany. Similar to SNOMED CT, LOINC is used by CDA documents as a vocabulary domain, encoding CDA components into a standard database of terms.

RxNorm: RxNorm [RxN07] is a standardized nomenclature for clinical drugs produced by the National Library of Medicine. A clinical drug is a pharmaceutical product administered to a patient with a therapeutic or diagnostic intent. The definition of a clinical drug combines its ingredients, strengths, and form. The form refers to the physical form in which the drug is administered in a prescription or order. For example, two possible definitions of clinical drugs are: (a) *Acetaminophen 500 MG Oral Tablet*, for a generic drug name, and (b) *Acetaminophen 500 MG Oral Tablet [Tylenol]*, for a branded drug name [RxN07].

The purpose of RxNorm is to standardize the information exchange both between systems within the same organization and between different organizations, allowing various systems using different drug nomenclature to share data efficiently. It is intended to cover all prescription medications approved for use in the United States. RxNorm is conformed by concepts, collections of names identical in meaning at a specified level of abstraction. Each concept can be mapped to different string values in different systems, all naming things that are the same. It also provides a linkage to terms from other vocabularies (i.e., the concept *Ortho-Novum 7/7/7 21 Tablets* is a term from the SNOMED vocabulary; it is not within RxNorm at all, except as it is related to RxNorm within the RXNREL table [RxN07]).

6.2.2 Clinical Document Architecture

The Clinical Document Architecture (CDA) is an XML-based document markup standard that specifies the structure and semantics of clinical documents, such as discharge summaries and progress notes, for the purpose of exchange. It is an American National Standards (ANSI) approved HL7 standard, intended to become the de facto electronic medical record.

According to the developers of CDA version 2.0 [CDA07], the main characteristics of the CDA standard are:

a. **Persistence:** The clinical documents exist in an unaltered state for a time period defined by local and regulatory requirements.

b. **Stewardship:** A clinical document is maintained by an organization entrusted with its care.

c. **Authentication:** The clinical records are intended to be legally authenticated.

d. **Context:** The clinical document specifies its own default context.

e. **Wholeness:** Authentication of a clinical document applies to the whole instance and the full context. Also, it is a complete and persistent set of information including text, images, sound and other multimedia content.

f. **Human readability:** A clinical document is human readable.

Some projects already implementing CDA are: Continuity of Care Record (USA) [AST07], SCIPHOX (Germany) [SCI07], MedEmed (Canada) [OBJ03], PICNIC (Denmark) [PIC07], e-Claims Supporting Document Architecture (Canada), Health Information Summaries (New Zealand), Aluetietojaerjestelmae (Finland) [IV02] and Dalhousie Discharge Summary System (Canada).

Figure 6.4 [DAB$^+$06] shows a fragment of the CDA's Object Model that represents the semantic constructs of the RIM, depicting the connection from a document section to a portion of the CDA clinical statement model with nested CDA entries.

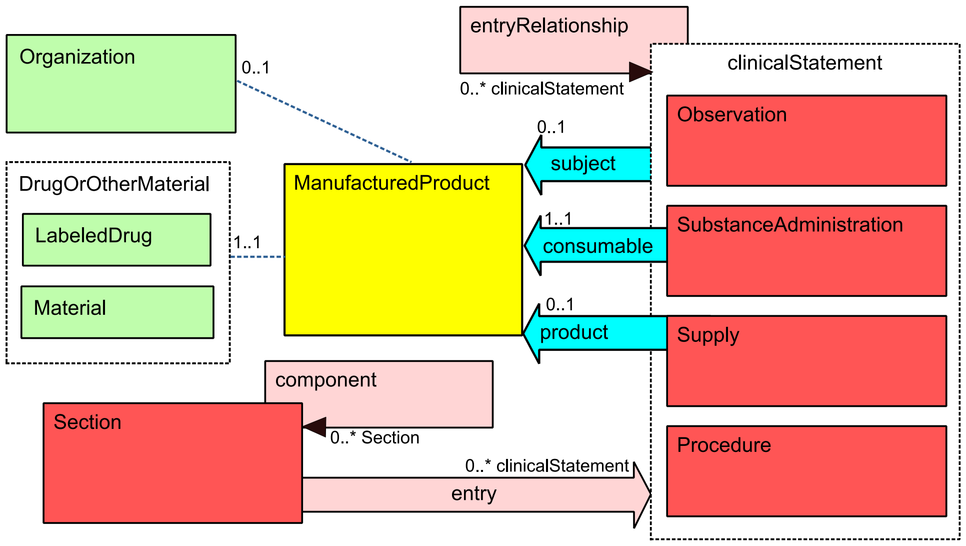


Figure 6.4: Fragment of CDA Object Model.

The colors in Figure 6.4 identify these classes with the core classes of RIM as depicted in [DAB$^+$06] (Red for Act specializations, blue for Participations, green for Entities, yellow for Roles and pink for Relationships). As described in [DAB$^+$06], an Act can have zero to many ActRelationships to other Acts, and can have zero to many Participations, each played by an Entity in some Role. A Role relates two Entities; the Entity playing the Role is represented by a solid line and the Entity who recognizes the role is represented with a dashed line. Thus, in Figure 6.4, a “legalAuthenticator” is a Participant of a “ClinicalDocument” Act and is played by a “Person” Entity in an “AssignedEntity” Role that is recognized by an “Organization” Entity [DAB$^+$06].

The “Component” class is an ActRelationship that may link the “ClinicalDocument” to the body choice (“NonXMLBody” or “StructuredBody”) or the “StructuredBody” to each nested “Section”. The “StructuredBody” contains one or more Section components, each of which contains a human readable title and a “narrative block”, the human readable content that has to be populated by the document originator and rendered by the recipient. Each section can also contain any number of CDA entries and external references. The CDA narrative block is wrapped by the “text” element within the “Section” element, and provides a slot for the human readable content needing to be rendered. Within a document section, the narrative block represents content to be rendered, whereas CDA entries represent structured content provided for a computer. CDA entries encode content present in the narrative block of the same section. The example shows two "Observation" CDA entries, although several other CDA entries are defined.

Figure 6.4 shows, at the right of the Section class, the Entry Relationship, which leads to the clinical statement portion. Each Entry represents structured content intended for computer processing such as decision support applications. Also, the clinicalStatement class contains specializations of the Act class (in this case Observation, SubstanceAdministration, Supply and Procedure) that will be included in the formal representation [DAB$^+$06].

CDA external references always occur within the context of a CDA entry, and are wrapped by the "reference" element. External references refer to things that exist outside the CDA document - such as some other image, some other procedure, or some other observation (which is wrapped by the "ExternalObservation" element). The CDA entry that wraps the external reference can be used to encode the specific portions of the external reference that are addressed in the narrative block.

Listing 9.1 in the Appendix depicts a sample CDA document D_1, which is wrapped by the "ClinicalDocument" element, as it appears in line 2 of this figure. The CDA header (lines 3-29) identifies and classifies the document, and provides information about authentication of the record as well as the participants (patient and involved providers). Figure 6.5 depicts the tree representation of D_1.

The CDA body (lines 31-82), which is wrapped by the "StructuredBody" element, is the core of the document and contains the clinical report. It can be either an unstructured segment or an XML fragment. We focus this study in the structured XML definition of the clinical report, which is the one providing the most opportunity for high-quality information discovery. Traditional Information Retrieval (IR) approaches [Sal89, BYRN99] can be applied to the unstructured scenario.

6.2.3 Information Discovery on General XML Documents

XML has emerged as the de facto standard format to represent and exchange data through the World Wide Web and other heterogeneous environments, spanning a wide variety of domains and

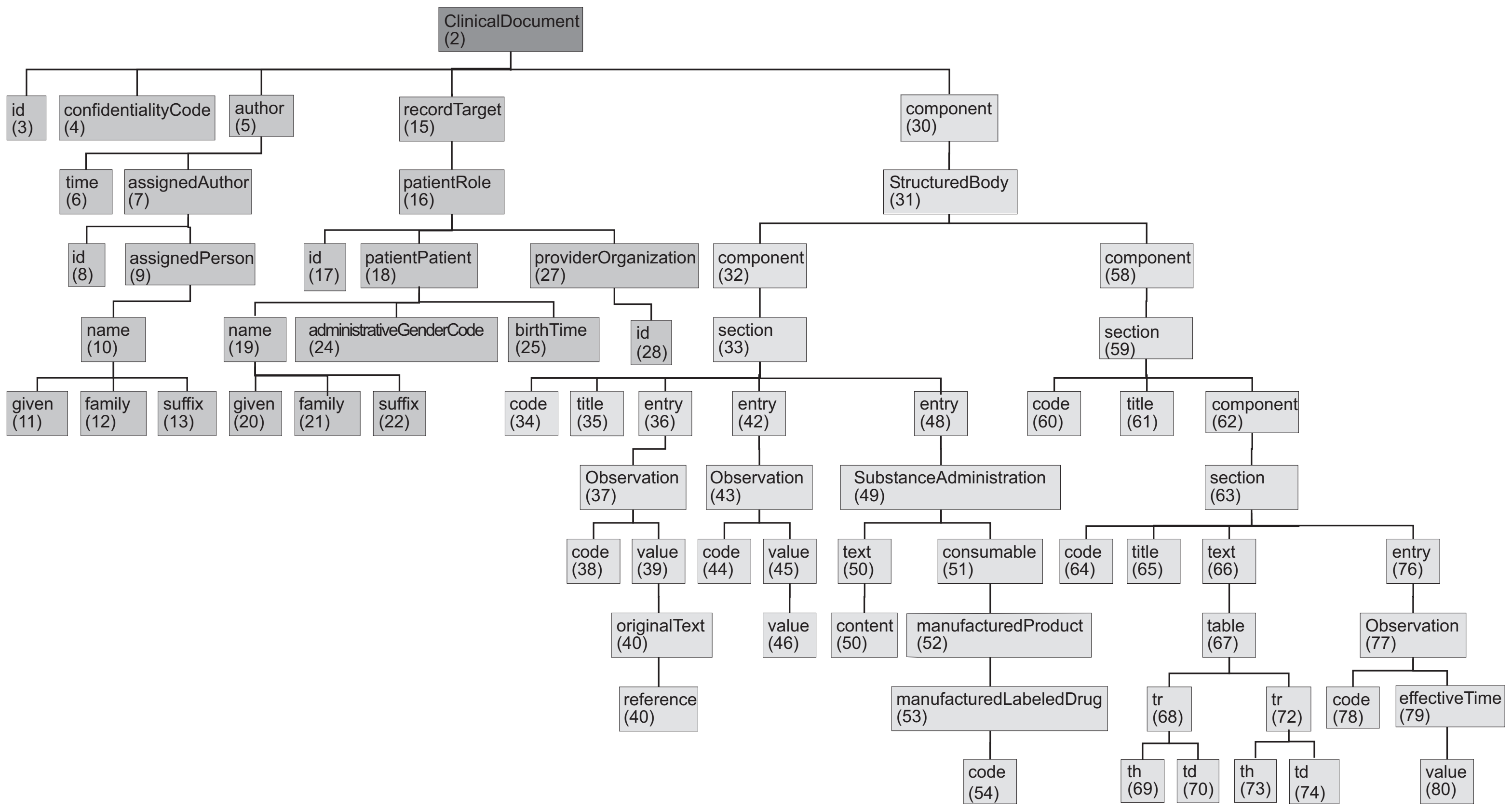


Figure 6.5: Tree representation of document D_1 in Figure 9.1

applications. The increased popularity of XML repositories and XML documents in general must be accompanied by effective ways to retrieve the information stored in this format. In this section we present an overview of previous work on searching XML documents. This corpus of work will be viewed as the starting point to present the challenges of information discovery on CDA XML documents in Section 6.3.

Limitations of Traditional Information Retrieval (IR) Methods: The traditional and popular text-based search engines cannot deal effectively with XML documents due to a series of limitations. First, text-based search engines do not exploit the XML tags and nested hierarchical structure of the XML documents. Second, the whole XML document is treated as an integral unit and is returned as a whole, which is unacceptable given the possibly large sizes of XML documents –in contrast we would like to be able to return parts of an XML document. A third drawback is the keyword proximity concept in XML, which can be measured in terms of containment edges, in contrast to the traditional keyword proximity search in text and HTML documents. That is, two keywords that may appear physically proximal in the XML file may be distant or unrelated in the tree-structured XML document and vice versa.

Previous Work on Searching XML documents: XRANK [GSBS03] computes rankings at the granularity of an element, considering element-to-element links in addition to document-to-document links. XRANK ranks the XML elements by generalizing the PageRank algorithm [BP98], combining the ranking of elements with keyword proximity.

XSEarch [CMKS03] ranks the results taking into consideration both the degrees of the semantic relationship and the relevance of the keyword. XSEarch also adds the power of distinguishing between tag names and textual content. They also disallow results where the same tag name appears more than once in nodes of a vertical result path. Cohen et al. [CKKS05] present an extended framework to specify the semantic relationship of XML elements, providing a variety of interconnection semantics based on the XML schema, improving the quality of the ranking of XSEarch. XIRQL [FG01] utilizes a different strategy to compute its ranking, defining index units, specific entity types that can be indexed and used for tf-idf computation.

Schema-free XQuery [LYJ04] refines the work of XSEarch by utilizing meaningful lowest common ancestors instead of the concept of interconnected nodes, skimming some unrelated, “too inclusive” elements that are not supposed to be returned. Cohen et al. [CKKS05] improve even further this approach by including the schema into the framework and discovering interconnection information. Xu and Papakonstantinou [XP05] define a result as a “smallest” tree, that is, a subtree that does not contain any subtree that also contains all keywords. Hristidis et al. [HP06] group structurally similar tree-results to avoid overwhelming the user.

Previous works define a query answer in several different ways. XRANK, XIRQL and TeX-Query [AYBS04] define an answer to be a document fragment (generally a subtree) –the most specific fragment of the XML document is typically the highest ranked answer. In contrast, XSEarch defines the result to a query to be a sequence of XML nodes and null values forming a path that connects the elements that contain the keywords or that satisfy the query predicates. On the other hand, Carmel et al. [CMM$^+$03] utilize XML Fragments as the syntax to specify the query but their query answers consist of entire documents, not fragments. Pradhan [Pra06] present a flexible algebraic approach for defining results’ properties in the query, in addition to a list of keywords.

XKeyword [HGP03] operates on an XML graph (with ID-IDREF edges) and returns a subtree of minimum size that contains all query keywords. The World Wide Web Consortium has proposed syntactic and semantic extensions to XQuery and XPath [AYBS04, XQu07] to support full-text search capabilities. Amer-Yahia et al. [AYCD06] present an Algebra to support such an extension.

6.3 Challenges of Information Discovery on CDA Documents

In this section we present a series of challenges that have to be addressed to effectively perform information discovery on a corpus of CDA documents. For simplicity we focus on plain keyword queries, although the same challenges are valid for semi-structured queries as well –a semi-structured query is a query where partial information about the structure of the results is provided. For example, specify that we are only interested in “code” elements under “Observation” elements.

We discuss why the general work on information discovery on XML documents (Section 6.2.3) is not adequate to provide quality information discovery on CDA XML documents. The key reasons are the complex and domain-specific semantics and the frequent references to external information sources like dictionaries and ontologies.

We use Document D_1 depicted in Figure 9.1 as our running example, along with the plain keyword queries of Table 6.2.

Table 6.2: CDA Document Queries

Id	Keyword Query
q_1	“Asthma Theophylline”
q_2	“Substance Theophylline”
q_3	“Respiratory Theophylline”
q_4	“Temperature”

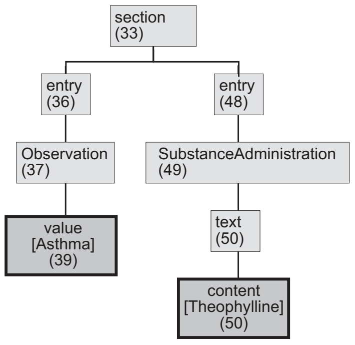
6.3.1 Structure and Scope of Results

In contrast to traditional Web search where whole HTML documents are returned as query results, in the case of XML documents and particularly CDA documents, we need to define what a meaningful query result is. Previous work has studied different approaches to define the structure of results. A corpus of works [AYBS04, FG01, GSBS03] consider a whole subtree as result, that is, a result is unambiguously defined by the lowest common ancestor (LCA) node of the keyword nodes. We refer to this approach as *subtree-as-result*. For example, XRANK favors deeply nested elements, returning the deepest node containing the keywords as the most specific one, having more context information. In contrast, a path as the result is proposed by [ACD02, BNH$^+$02, HP02, CMKS03, HPB03]; where a minimal path of XML nodes is returned that collectively contain all the query keywords. Note that we use the term “path” loosely to differentiate it from the subtree-as-result approach, because it can be a collection of meeting paths (a tree) for more than two query keywords. We refer to this approach as *path-as-result*.

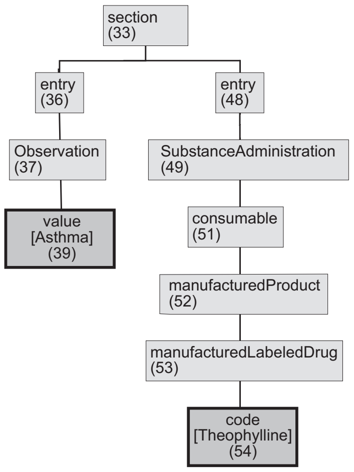
Example: *To illustrate this challenge we execute query q_1 on document D_1. For the path-as-result approach there are two candidate results depicted in Figures 6.6 (a) and (b) because of the two appearances of the keyword “Theophylline” in lines 50 and 54. For the subtree-as-result approach, only the subtree rooted at the XML node of line 33 is a possible result.*

It is unclear whether the subtree-as-result or the path-as-result is a better fit for searching CDA documents. The discussion on minimal information unit below sheds more light to this aspect.

Another issue is the *scope* of a result, in particular, whether results spanning across EMRs should be produced. For instance, two query keywords may be found on two EMRs authored by the same doctor (the doctor becomes the connection element as discussed in Section 6.3.10. If the query is "drug-A drug-B death" then clearly two-EMR results are not useful since if different patients took the two drugs no correlation between the drugs can be drawn. On the other hand, if the query is "rare-disease-A rare-disease-B" then it may be useful to find a doctor who has treated two patients that have had one disease each. A simple solution to this dilemma is to allow the user to explicitly specify if cross-EMR results are allowed.



(a) Path connecting "Asthma" in line 39 and "Theophylline" in line 50



(b) Path connecting "Asthma" in line 39 and "Theophylline" in line 54

Figure 6.6: Atomic path results for Query q_1. The highlighted nodes match the terms.

Finally, doctors would like to be able to specify the results' schema in some cases, which in turn limits the types of elements searched for the query keywords.

6.3.2 Minimal Information Unit (MIU)

It is challenging to define the granularity of a piece of information in a way that it is self-contained and meaningful, but at the same time specific. For example, in Document D_1 returning the "value" element of line 45 without the preceding "code" element is not meaningful for the user. Hence, the "value" element is not an appropriate MIU, whereas the enclosing "Observation" element could be.

Furthermore, for some queries it is required to include into the result some elements that do not contribute in connecting the query keywords or are part of the MIU of such a connecting node. For instance, the "patientPatient" element should be included in the result of q_1 if a practitioner submits the query, but not if a researcher does. Such personalization issues are further discussed in Section 6.3.14.

Another issue is the static definition of MIU. In XKeyword [HPB03], a "target object" is the equivalent of an MIU and they are defined statically on the schema by a domain expert. Xu et al. [XLWS06] also define MIUs in a static manner. Such static MIU definitions are not adequate for CDA information discovery, as the following scenario explains. For the query "Body height" a reasonable result is the "Observation" element in lines 77-81. On the other hand, for the query "1.77" this same element is not meaningful since obviously the user knows that "1.77" is a height value, but the patient who has this height is probably of more interest. Hence, there is a need to dynamically specify MIUs.

Example: *The tight semantic relationship between the nodes in the subtree rooted at the element "SubstanceAdministration" in line 49 of Figure 9.1 can lead the system expert to consider this subtree as a MIU. In this case, the single result of query q_1 on Document D_1 for the path-as-result approach is the one shown in Figure 6.7. If, in contrast, every element in the tree is considered a minimal information unit, then the two paths depicted in Figure 6.6 are the results for this query.*

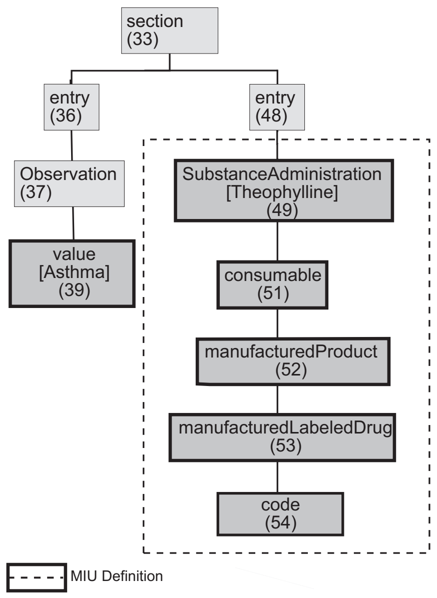


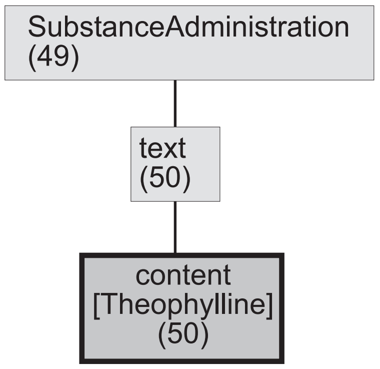
Figure 6.7: Result for q_1 using *SubstanceAdministration* as Minimal Information Unit

6.3.3 Semantics of Node and Edge Types

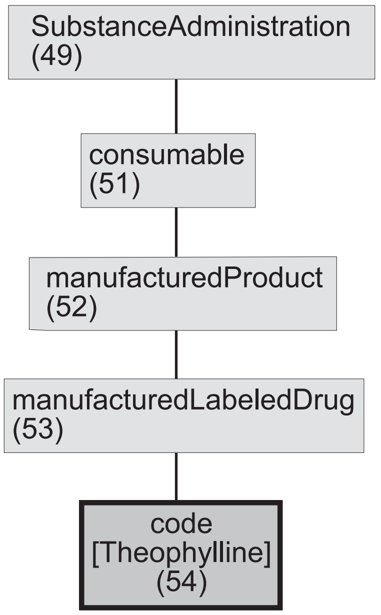
It is challenging to incorporate the rich semantic information available for the clinical domain, and particularly for the elements of a CDA document, in the results' ranking process. At the most basic, a domain expert statically assigns a weight to each node and edge type, as in BANKS [BNH$^+$02]. In addition to that, we can assign a relevance to whole paths on the schema as explained below. Furthermore, it is desirable that the degrees of semantic association are adjusted dynamically exploiting relevance feedback [SB97] and learning [Mit97] techniques.

The equivalent of a schema for a CDA document is the CDA Release 2 Object Model (Figure 6.4), showing the connection from a document section to a portion of the CDA clinical statement model [DAB$^+$06]. Edge and node weights can be specified on this Object Model. For example, the relationship between a substance and the patient it was prescribed to may be more relevant than the relationship between the substance and the doctor who prepared the EMR.

As mentioned above, assigning relevance degrees to whole paths instead of single edges can improve the ranking quality. For example, the path "$SubstanceAdministration \rightarrow consumable \rightarrow manufacturedProduct \rightarrow manufacturedLabeledDrug \rightarrow code$" could have a higher or equal weight than "$SubstanceAdministration \rightarrow consumable \rightarrow manufacturedProduct$".



(a) Path connecting "Substance" in line 49 and "Theophylline" in line 50



(b) Path connecting "Substance" in line 49 and "Theophylline" in line 54

Figure 6.8: Path Results to q_2.

This is particularly important for cases where a syntactically long path corresponds to a semantically tight association. For instance, the path "$SubstanceAdministration \rightarrow consumable \rightarrow manufacturedProduct \rightarrow manufacturedLabeledDrug \rightarrow code$" in lines 49-57 of Figure 9.1 has four edges, but intuitively this sequence of elements will typically appear as an indivisible unit. Hence, this path may be viewed as a single edge for the purpose of ranking. In general, the information discovery algorithm must neutralize the effect of the schema design decisions of CDA by considering a semantic instead of a syntactic distance.

Example: *Consider query q_2 executed over D_1. We can see with this query the need to index and query the XML tags in addition to the values; in this case the keyword "Substance" matches the tag "SubstanceAdministration" in line 49. Figure 6.6 shows two possible results to q_2. Even though the first result only involves two edges (whereas the second involves four), it could be that the second result is ranked higher if the path "$SubstanceAdministration{\rightarrow}consumable{\rightarrow}manufacturedProduct{\rightarrow}manufacturedLabeledDrug{\rightarrow}code$" is viewed as a single edge.*

6.3.4 Access to Dictionaries and Ontologies

CDA documents routinely contain references to external dictionary and ontology sources through numeric codes. As an example, document D_1 includes references to LOINC [Log06] and SNOMED CT [SNO08] in lines 34 and 38 respectively. Hence, it is no longer enough to answer a query considering the CDA document in isolation, as is done by all previous work on information discovery on XML documents (Section 6.2.3). In this setting, the query keywords may refer to text in the CDA document or an ontology that is connected to the CDA document through a code reference. For example, the query keyword "appendicitis" may not be present in the document but its code might be present, so we need to go to the ontology and search for the query keyword there.

On a high level, it would be desirable to view the data graph (the CDA document) along with the ontology graph (e.g., SNOMED) as a single "merged graph". An approach to achieve that is the following:

a. View a code node in a CDA document and the corresponding ontology node as a single node, that is, collapse these two nodes. Equivalently, add an edge with infinite weight between them (assuming higher weight denotes higher association).

b. For free text nodes (with no code) v of the CDA document we add an edge between v and each ontology node u with weight equal to the IR similarity between the content of v and u. Only the edges with weight greater than the specified threshold are finally created.

This second technique can be omitted if we assume that the author of the CDA document is including the ontology/dictionary codes where appropriate and there are matching ontology entities for all real entities in the CDA document.

An alternative technique has been described to incorporate ontology information in the query processing [HHP06]. Designed to enable keyword search on data graphs with authority flow semantics, the ObjectRank authority flow algorithm [BHP04] is executed on the ontology graph to rank the ontology nodes with respect to the query, and then uses the terms of the top-ranked ontology nodes to expand the original query.

Example: *Query q_3 executed on D_1 would have an empty result (for AND semantics) if the ontologies/dictionaries were not text searched. However, if the intuition discussed above is applied, the same results as in q_1 are valid, since the query term "respiratory" is associated to the term "Asthma" in D_1 through relationships of the SNOMED ontology, as shown in Figure 6.2.*

Note that it is challenging to rank results produced by exploiting ontological relationships as discussed in Section 6.3.5.

Performance: The solutions proposed to exploit ontology/dictionary information incur challenging performance issues. Two high-level techniques that can be employed to realize the above query semantics are:

a. Search all ontologies for the query keywords, find adequately associated codes, and then search the CDA documents for these codes.

b. Start searching the documents and for each ontology code encountered, lookup the keywords in the corresponding ontology.

Furthermore, it is challenging to develop efficient pre-computation and runtime algorithms to facilitate the expensive in terms of execution semantics of the merged data and ontologies graph discussed above.

Another performance challenge arises due to the size of the ontologies. As mentioned in Section 6.2.1, SNOMED CT contains more that 235,000 concepts and 1,200,000 relationships between them. This corresponds to more than 2GB of compressed data, which will play a role in deciding which execution approach will be more efficient.

6.3.5 Different Types of Relations in Ontology

We need to assign an appropriate value to each of the relations present in the ontologies. SNOMED CT, for example, has four different types of relationships:

1. Defining characteristics,
2. Qualifying characteristics,

3. Historical relationships and

4. Other relationships.

Figures 6.2 and 6.3 include relations such as “May be”, “Finding site of” and “Has finding site” in addition to the most common “Is a” relationship. Stricter and stronger relations in the ontology should intuitively have a higher weight.

Furthermore, we need to take into consideration the direction of the edges. For instance, following “Is A” edges specializes and restricts the search on the one direction, but generalize in the other direction, with the risk of returning imprecise terms.

We must also consider the number of incoming and outgoing edges that each node has. For example, some SNOMED CT concepts such as “Duplicate concept” or “Invalid concept” participate in historical relationships and possess a large incoming degree. Navigating these historical relationships to concepts with such large in-degrees may not be beneficial to the information discovery process.

A possible approach to measure the degree of association between nodes of an ontology graph is to execute ObjectRank [BHP04] on the ontology graph, as described by Hwang et al. [HHP06]. In particular, for query q_3 we can place the nodes containing the keyword “Respiratory” in the base set and then execute ObjectRank. If the node containing the term “Asthma” (line 39 of D_1) ends up having a higher score than the node containing the term “Bronchitis” (line 45 of D_1), then the “Asthma” node will be preferred. This process can be further improved by assigning different authority transfer bounds [BHP04] to various edge (relationship) types of the ontology according to their semantic association.

Example: *As an example we execute query q_3 on D_1. We can see in the ontology graph of Figure 6.2 that “Asthmatic Bronchitis” and “Asthma” are both related to “Respiratory”, but “Asthmatic Bronchitis” is two “Is A” edges away from “Respiratory”, whereas “Asthma” is only one edge away. Hence a result containing “Theophiline” and “Asthma” (line 39) would be better than one containing “Theophyline” and “Bronchitis” (line 45).*

6.3.6 Arbitrary Levels of Nesting

We can find an arbitrary number of levels of nesting and recursion in the definition of components and sections, as exemplified in the path $component \rightarrow section \rightarrow component \rightarrow section$ in lines 58-63 of Figure 9.1.

Taking into consideration the semantics of the document, the interconnection relationship rule of XSEarch [CMKS03], where the same tag may not appear twice in internal nodes of a result path, cannot be applied since the same tag can appear twice in a vertical path (top-to-bottom). In particular, the rule of XSEarch assumes that a vertical path may not contain the same tag twice, since elements with the same tag name are typically in the same level of the tree. This is clearly not true for CDA documents.

Hence, the XSEarch interconnection relationship should be modified considering semantic information of the surrounding elements. For instance, if we assume that a "component" element represents a hospitalization, then if two keywords with the same tag appear in different components of the same section, the XSEarch rule can be applied, but not if they are in two different sections of same component.

6.3.7 Handling ID-IDREF Edges

CDA entries can include pointers to "content" elements of the CDA Narrative Block; similarly, "renderMultiMedia" elements of the CDA Narrative Block can point out to CDA entries. The "content" element can contain an optional ID attribute to identify it, and it can serve as the target for a reference. The "originalText" component of a RIM attribute can then refer to this identifier, indicating the original text. As an example we can find an ID attribute in line 50 of Figure 9.1. A reference to this element is found in the "originalText" element of line 40.

These edge types have been ignored for results computation by previous search strategies like XRANK, which only utilizes the hyperlinks (ID-IDREFs) for score calculation. That is, results are always subtrees ignoring the ID-IDREF edges. We want to exploit these edges in producing the results. A consequence of this issue is the fact that the result can be a graph (with cycles) and not a tree. In this case, we need to decide whether we break the cycles to return a tree as the answer, since a tree is typically easier to present and reason about. Also, similar to XRANK, ID-IDREF and containment edges could be assigned different weights.

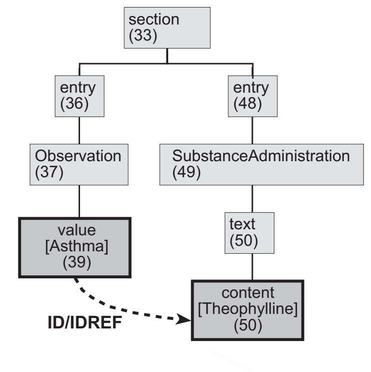


Figure 6.9: Result to Query q_1 considering ID/IDREFS.

Example: *We execute query q_1 on the sample document D_1. We obtain the two path results depicted in Figure 6.6, but if we include the ID-IDREF hyperlink between elements in lines 40 and 50 of Figure 9.1 we obtain the graph depicted in Figure 6.9, containing a cycle.*

In case we decide the best solution is to break the cycles, the next issue is to decide the best edge to remove. The simplest possibility is to eliminate the hyperlink and preserve a path as the one shown in Figure 6.6(a). Alternatively, the weights and directions of the edges may be taken into account.

6.3.8 Free Text Embedded in CDA Document

In some cases, plain text descriptions are added to certain sections to enrich the information about the record or to express a real life property not codified in dictionaries or ontologies. As a first measure, traditional text-based Information Retrieval techniques [Sal89, BYRN99] should be included in the architecture to support such cases.

Another technique to address the coexistence of semi-structured and unstructured data is presented in [HGP03], where IR and proximity rankings are combined.

In addition to embedded plain text, HTML fragments can also be included to the CDA document, resulting in a mix of semantic mappings. For instance, line 50 in Figure 9.1 describes the

Listing 6.1: Free text occurence of keywords on query q_4.

```
<text><content ID="m1">Theophylline</content>20 mg every
    other day, alternating with 18 mg every other day.
  Stop if temperature is above 103F.</text>
```

Listing 6.2: Embeded HTML fragment is the result of query q_4.

```
<th>Temperature</th>
```

full-text description of the dosage for a substance. Due to the complex nature of this description, there is no single entity in the ontology to accurately match it.

Example: *To exemplify this challenge we execute query q_4 on our sample document D_1. Listings 6.1 and 6.2 show two possible results for this query assuming each element is a MIU. Listing 6.1 presents a free-text entry containing the keyword "Temperature", whereas Listing 6.2 depicts an HTML fragment also containing the keyword. Without additional semantic information, these results cannot be ranked based on their structure; appropriate IR techniques should be applied to solve this challenge. For instance, the second result may be ranked higher since it has a smaller document length (DL).*

6.3.9 Special Treatment of Time and Location Attributes

After discussing with medical researchers and practitioners, we found that time and location are critical attributes in most queries. For instance, for the query "drug-A drug-B" the doctor is probably looking for any conflict between these drugs, and hence the time distance between the prescriptions of these drugs for a patient is a critical piece of information. Location is also important since two patients located in nearby beds in the hospital should be viewed as associated because infections tend to transmit to neighboring beds. Clearly, it is challenging to standardize the representation of such location information within an EMR.

Furthermore, time and location can lead to the definition of metrics similar to the inverse document frequency (idf) in Information Retrieval [Sal89]. For instance, asthma is more common in summer; hence a patient who has asthma in winter should be ranked higher for the query "asthma". Similarly, a patient who has the flu in a town where no one else has it should be ranked higher for the query "flu". These associations are too complex since time can be used to define time, distance, or periodicity. Similarly, location relationships can be specified either within a hospital or across towns.

Finally, there should be a way to specify time intervals in the query, possibly using a calendar interface, and then use the specified time window as an answers filter. Specifying the time-distance between the keywords can also be useful. For instance, the query "newborn heart block" which is often needed at Miami Childrens Hospital, should not return a patient who got a heart block when he was 60 years old but the word "newborn" appeared in his EMR in a description field of her birth day.

6.3.10 Identity Reconciliation and Value Edges

A single real-life entity (e.g., a medication or a doctor) is duplicated every time it is used in a CDA. Hence, associating two records of the same author, or two patients with the same medication is hard. In contrast, in previous work on searching XML documents, a real-life entity is typically represented by a single XML element, which is linked using ID-IDREF edges where needed. For instance, in XKeyword two articles of the same author have an IDREF to the same author element.

The problem of reference reconciliation has been tackled both in the context of structured databases [DHM05, HS95, MNU00, MW03, SB02, TKM02, Win95] and in the context of free text document collections [ML95, MW03, NC01, ZAR02]. However, focusing on the domain of CDA documents allows manually specifying rules by a domain expert on what types of elements are good candidates for referencing identical real-life objects, in case these elements have identical or similar values.

In particular, we can identify on the schema the elements that have the property that the same value probably means the same real-life entity, so that "value edges" can be added accordingly. Such elements may be the "assignedAuthor", the "patientPatient", the "manufacturedLabeled-Drug" and so on. On the other hand, no "value edge" should be added between two "title" elements. E.g., two patients who both have “Physical Examination” value on the “title” element (line 61 in Figure 9.1) are not related in any way.

As another example, if two medications have the same SNODEM code, they should be associated. However, if a drug and its generic have different SNOMED codes, such associations are hard to establish.

Another challenge involves the use of multiple possibly overlapping ontologies across the corpus of CDA documents. For instance, different codes are used for the term ”Asthma” in SNOMED CT and LOINC (195967001 and 45669-9 respectively). Ontology mapping techniques can be leveraged [DHM05, HS95, ML95, MNU00, MW03, NC01, SB02, TKM02, Win95, ZAR02] (for more details on such techniques see Section 6.4). Furthermore, we can probabilistically extend these initial mappings using “meta-rules” like the following [MNJ04]: if two concepts C_1 and C'_1 match, and there is a relationship q between C_1 and C_2 in Ontology O and a matching relationship q' between C'_1 and C'_2 in Ontology O', then we can increase the probability of match between C_2 and C'_2. Hence, code elements in a single or multiple CDA documents that refer to the same or similar real-life entities will be associated through a “value edge”.

6.3.11 EMR Document-as-Query

An alternative query type to the plain keyword query is using a whole (or part of) EMR (CDA) document as the query. This approach can be used in order to find similar CDA documents, that is, CDA documents of patients with similar history, demographic information, treatments, and so on. The user should be able to customize and personalize such an information discovery tool to fit her needs. For instance, a researcher may not consider the physicians (author of CDA document) name when matching CDA documents, and could specify that a generic medication should be viewed as identical to the non-generic equivalent. Previous work on document content similarity [And00] and XML document structural similarity [NJ02] can be leveraged to solve this problem. The latter corpus of works is based on the concept of tree edit distance. The best known algorithm for computing tree edit distance between two ordered trees is by Zhang and Shasha [ZS89] with the time complexity of roughly $O(n^4)$ where n is the number of the nodes in a tree. Chakaravarthy at al. [CGRM06] match pieces of unstructured documents to structured entities, whereas we want to match a structured document to other structured or unstructured documents.

Furthermore, such document-as-query queries can be used to locate medical literature relevant to the current patient. In this scenario, the EMR application could have a button named "relevant literature" that invokes an information discovery algorithm on PubMed or other medical sources. Price et al. [PHOE02] present a first attempt towards this direction, where they extract all MeSH terms (MeSH refers to the U.S. National Library of Medicine's controlled vocabulary used for indexing articles for MEDLINE/PubMed) from an EMR (not specific to CDA) and then query MEDLINE using these terms. The structured format of CDA documents can potentially allow more elaborate searching algorithms where multiple terms that are structurally correlated can construct a single and more focused query on medical literature sources.

6.3.12 Handle Negative Statements

A substantial fraction of the clinical observations entered into patient records are expressed by means of negation. Elkin et al. [EBB$^+$05] found SNOMED-CT to provide coverage for 14,792 concepts in 41 health records from Johns Hopkins University, of which 1,823 (12.3%) were identified as negative by human review. This is because negative findings are as important as positive ones for accurate medical decision making. It is common in a medical document to list all the diagnoses that have been ruled out, e.g., state that "the patient does not have hypertension, gout, or diabetes". This creates a major problem when searching medical documents. Today, one has to examine the terms preceding a diagnosis to determine if this diagnosis was excluded or not. Ceusters and Smith [CS05] propose new ontological relationships to express "negative findings". It is challenging to handle such negative statements for an information discovery query in a way that the user can specify whether negated concepts should be excluded or not from the search process.

6.3.13 Handle Extension Elements

Locally defined markup can be used to extend CDA when local semantics have no corresponding representation in the CDA specification. Such user- or institution-defined element types are hard to incorporate to the global semantic information, since it is not possible to define general structural requirements for the results, as in XSEarch [CMKS03] and the work of Xu and Papakonstantinou [XP05].

6.3.14 Personalization

The information discovery engine should provide personalized results depending on the preferences of each individual user. For example, for different doctors, different entities and relationships in the CDA components are more important. For some healthcare providers, the medication may be more relevant than the observation, or the medication may be more relevant than the doctor name. Also the relationships in ontologies may be viewed differently.

Furthermore, depending on whether a user is a nurse, a pharmacist, a technician or a physician, the system could automatically assign different weights on edges and nodes of the CDA Object Model (Figure 6.4) to facilitate the information needs of the users.

6.3.15 Confidentiality of Records

The level of confidentiality of the medical record is indicated by the *confidentialityCode* element in the header section of the record, taking the values “normal”, “restricted” and “very restricted”. The value of this element, shown in line 4 of Figure 9.1, may dictate at what level we may return results for an executed query. If *confidentialityCode* is set to “restricted” but no personal info is contained in the result, then the result could be output. Otherwise, the credentials of the user should also be taken into consideration to validate whether the user has the right privileges to obtain the query results.

As mentioned in Section 6.2.1, LOINC codes are already part of HIPAA [Hea08], complying with the confidentiality standards imposed by the Federal Government on the Insurance and Health Care Industries.

6.4 Related Work on Information Discovery on Electronic Health Records

This section reviews some research areas that are related to the problem we are introducing in this chapter, in addition to the XML information discovery techniques reviewed in Section 6.2.3: the testing and evaluation of IR techniques on XML, the problem of automatic ontology mapping, and the limitations of medical ontologies.

To test and evaluate IR techniques on XML documents, the INitiative for the Evaluation of XML Retrieval (INEX) [INi09, FGKL02] was created in 2002 to provide the infrastructure and means to evaluate the retrieval methods and techniques and to compare results, specifically providing a large XML test collection and appropriate scoring methods, for the evaluation of content-oriented XML retrieval systems. For INEX 2007, the test collection consists of more than 650,000 XML-encoded articles from the Wikipedia project, compiling 4.6 Gigabytes of textual information. These documents are organized in topics, with relevance assessments defined for each topic. A series of content-only (CO) and content-and-structure (CAS) queries is defined for each topic. The CO queries resemble those used in the Text REtrieval Conference (TREC) [Tex07].

Even when representing the same domain, information sources may be of heterogeneous semantics, resulting in a necessary mapping between ontologies and schemata in order to compose the information and enable interoperation. This has been a research topic in recent years, providing strategies to compose different and heterogeneous sources, aiming to reduce the impreciseness and errors in such mappings. A large number of articles are listed at [Ont07]. ONION [WD01] and Prompt [NM03] use a combination of interactive specifications of mappings and heuristics to propose potential mappings. GLUE [DMDH02] employs machine-learning techniques to discover the mappings. OMEN [MNJ04] exploits schema-level information by using a set of meta-rules.

In recent years, one of the hottest research directions in medical informatics has been to address the biomedical terminology problem. Ontologies and description logics have been chosen to tackle this challenge, proving to be an adequate solution. But it has also been shown that description logics alone cannot prevent incorrect representations of the medical terminology, since frequently they are not accompanied of the proper theory to describe them. The inappropriate adoption of the UMLS Metathesaurus [UML07] has been specifically criticized and questioned in [CSF03], which cites these three problems: (1) There is a wide range of granularity of terms in different vocabularies. (2) The Metathesaurus itself has no unifying hierarchy, so you cannot take advantage of hierarchical relations. (3) There may be other features of vocabularies that get lost in their ‘homogenization’ upon being entered into the Metathesaurus. Hahn et al. [HRS99] recognize the value of biomedical terminologies as the starting point for an engineering-oriented definition of medical ontologies, in which the reviewing of concept consistency and hierarchy concludes with the inclusion of missing terms and the correction of misclassified concepts. A new approach has been proposed by [SAL$^+$07], in which they introduce a new level of abstraction to represent a match between a text fragment and an ontology; they facilitate the discovery of medical knowledge by adding semantic annotations (with domain knowledge from the ontology) to the syntactic parse trees from the processed documents.

6.5 Concluding Remarks

We have introduced the problem of Information Discovery on Electronic Medical Records (EMR), enumerating a series of challenges that must be addressed to provide a quality information discovery service on EMRs, specifically on Clinical Document Architecture (CDA) documents. The challenges are related to the semantics of the architecture, the XML definitions of CDA documents, and the convergence of the narrative structure associated with ontologies and dictionaries. More research is needed to address the ability of keyword searches to return meaningful results on CDA documents containing time-dependent relationships. Guidance is also needed in determining how ontologies can be best used in CDA documents to improve keyword search effectiveness and minimize information discovery times. We hope that this work will spur new research on this topic, which can have a dramatic impact on the quality of healthcare.

CHAPTER 7

ONTOLOGY-AWARE SEARCH OF ELECTRONIC HEALTH RECORDS

7.1 Motivation

The National Health Information Network (NHIN) and its data-sharing building blocks, RHIOs (Regional Health Information Organizations), are encouraging the widespread adoption of Electronic Medical Records (EMR) for all hospitals within five years. A key component of this effort is the standardization of EMR. To date, there has been little or no effort to define methods or approaches to search such documents effectively.

One of the most promising standards for EMR manipulation and exchange is Health Level 7's [HL708a] Clinical Document Architecture (CDA) [CDA07], which leverages a semi-structured (XML) format, and ontologies to specify the structure and semantics of EMRs for the purpose of Electronic Data Interchange (EDI).

In this chapter we present the XOntoRank system, which addresses the problem of facilitating ontology-aware information discovery within a corpus of XML-based EMR documents. By information discovery [PB99, HP02] we mean the extraction of relevant pieces of data from a database given a user query. Information discovery can be viewed as an extension of traditional Information Retrieval (IR), which ranks the relevance of unstructured documents given a keyword query. Hence, given a question (query) and a set of EMRs, we need to find the entities (typically subtrees) that match the query, and rank them according to their "goodness" with respect to the query. The success of Web search engines has shown that keyword queries are a useful and intuitive approach to information discovery. Therefore, we focus on keyword queries in this paper.

A large corpus of work (e.g. [FG01, GSBS03, CMKS03, HPB03]) addresses keyword search of XML documents, where the query keywords are matched to XML nodes and a minimal tree containing these nodes is returned. A variety of ranking techniques are used, ranging from the size of the result-trees to adaptations of Information Retrieval (IR) scoring. Investigators have explored ontologies (e.g. [KK05, STW05]) for XML querying; we compare them to our work in Section 7.7.

For example, consider the query *"Bronchial Structure Theophylline"* and a CDA document such as the one in Figure 9.1 in the Appendix, which is explained in detail in Section 6.2. The phrase *"Bronchial Structure"* does not appear in this document. Hence, most traditional XML-based keyword search systems will not return any results. However, this document contains an ontological reference to an *"Asthma"* concept defined in SNOMED (in Line 39, Figure 9.1). The SNOMED ontology further defines a *"finding-site-of"* relationship between *"Asthma"* and *"Bronchial Structure"* (as shown in Figure 6.2 in Section 6.2). Hence, based on the definitions in the ontology, a result tree connecting the *"Asthma"* node of Line 39 and the *"Theophylline"* node of Line 50 can be created as output.

The use of ontological definitions allows us to perform semantic search on the XML documents. We no longer require an exact match between keywords in the query and in the document, but we can make use of the domain ontology to infer a semantic relationship between keywords in the query and terms in the document. This allows returning more results than would otherwise be returned with an exact-match requirement. This paper makes the following contributions:

1. Introduce the problem of ontology-aware keyword search among XML-based EMR documents, which can be extended to general XML documents.
2. Define the semantics of what constitutes a result and how the results are ranked for the problem of ontology-aware keyword search within the EMR. We leverage previous work related to searching XML data.
3. Develop a set of techniques to compute the degree of association between ontological concepts that take into account both taxonomic *is-a* links as well as more general semantic relationships between concepts. This is a core component of our ranking framework.
4. Create and experimentally evaluate algorithms to answer efficiently ontology-aware keyword queries in EMRs. These algorithms were tested with real EMR data acquired from a local hospital.

We note that our study does not address the important privacy issues involved in accessing patient information, as required by HIPAA [Hea08]. The policies and principles described in [LAE$^+$04] could work as a starting point in achieving Hippocratic information discovery.

The rest of this chapter is organized as follows: Section 7.2 defines the problem and its semantics. Alternative approaches to compute the semantic relevance of an ontological concept to a keyword are presented in Section 7.3. In Section 7.4 we present the architecture. Section 7.5 presents the algorithms to implement the approaches of Section 7.3. Section 7.6 presents the experimental evaluation of XOntoRank. Section 7.7 presents previous work and we conclude in Section 9.

Notice that the relevant background for this chapter can be found in Section 6.2.

7.2 Problem Definition and Semantics

XML data: Our data collection is a set $D = \{T_1, \ldots, T_n\}$ of XML documents. We view an XML document as a labeled tree T. Each node $v \in T$ has:

a. A textual description $v.text$, which is the concatenation of its tag name, attribute names and values, and text content, and

b. An optional ontological reference $v.onto$, which typically consists of an integer code $v.onto.system$ for the referenced ontological system (e.g., SNOMED) and an integer code $v.onto.concept$ for the specific concept (e.g., *"Asthma"*).

Nodes with ontological reference are called *code nodes*. The set of ontological systems referenced by nodes in D is called ontological systems collection $O = \{O_1, \ldots, O_s\}$.

For instance, the node of Line 39 in Figure 9.1 has *v.text="value xsi:type="CD" code="195967001" codeSystem="2.16.840.1.113883.6.96" codeSystemName="SNOMED CT" displayName="Asthma"*, $v.onto.system = 2.16.840.1.113883.6.96$, and $v.onto.concept = 195967001$. Note that some attribute values like code strings are not included in $v.text$ since these are unlikely to be used in a query keyword or in ontology reference words from. An expert specifies the attributes that should not be included in the textual description.

In the algorithms presented in this paper we ignore ID-IDREF edges as well as inter-document references, since we build on tree search algorithms. However, the techniques we use to incorporate ontological information are straightforwardly applicable to graph search algorithms as well (i. e. when ID-IDREF edges are considered [HPB03]).

Keyword Search: A keyword query Q is a set $\{w_1, \ldots, w_m\}$ of keywords. Previous work, which ignores ontological references, has generally defined the results as subtrees of the XML documents that contain all query keywords (see Section 7.7 for an overview of related work). In this work we adopt the result semantics of XRANK [GSBS03], which is a popular representative of this class of works, and extend it to account for ontological references. Any other system could be extended in a similar way. The key extension is that instead of requiring keywords to be contained in the nodes of the result subtree, we require that the result subtree has nodes *associated* with every query keyword. Let $NS(v, w)$ (Node Score), whose computation is explained later, be the association degree of a node v with respect to a keyword w which is directly contained in v or is associated to v through an ontology. The result of Q for a document $T \in D$ is defined as follows. Let $R_0 = \{v | v \in T \wedge \forall w \in Q \exists u \in (Desc(v) \cup v)(NS(u, w) > 0)\}$ be the set of elements that are, themselves or through their descendant nodes, associated to all query keywords of Q. $Desc(v)$ is the set of descendants of v in T.

The result of the query Q is defined as:

$$Result(Q) = \{v | \forall w \in Q, \exists u \in (Desc(v) \cup v)(NS(u, w) > 0 \wedge \neg \exists t \in Desc(v)(t \in R_0))\} \tag{7.1}$$

Intuitively, a result v is an element that has sub-elements associated with each of the query keywords, but no sub-element is associated with all keywords. Note that $Result(Q)$ is a subset of R_0. The latter condition ensures we do not generate non-specific results.

For instance, if query *q=[“asthma”, “medication”]* is executed on the document of Figure 9.1, we get the XML fragment depicted in Figure 7.1, being the most specific sub-element in the CDA document that contains both terms in the query. Note that in the case, both terms are actually contained in the XML fragment. In general, though, the terms need not be in the fragment, but may be associated with nodes in the fragment through the ontology.

Listing 7.1: XML Fragment representing the answer to query *q=["asthma", "medications"]*

```
<Observation>
  <code code="84100007" codeSystem="
     2.16.840.1.113883.6.96"
      codeSystemName="SNOMED CT" displayName="
         Medications"/>
  <value xsi:type="CD" code="195967001" codeSystem=
      "2.16.840.1.113883.6.96" codeSystemName="SNOMED CT
         "
      displayName="Asthma">
    <originalText>
      <reference value="m1"/>
    </originalText>
  </value>
</Observation>
```

Score of results: As mentioned above, $NS(v, w)$ is non-zero if a node v directly contains w or is associated to w through an ontological system. This score is propagated to other nodes of the XML document as follows. The *propagated score* $PS(v, w, u)$ of an element v with respect to keyword w, assuming that a sub-element u of v has $NS(u, w) > 0$, is

$$PS(v, w, u) = decay^l \cdot NS(u, w) \tag{7.2}$$

where $l = distance(v, u)$ is the number of containment edges between v and u. $Decay$ is set between 0 and 1 to account for the specificity of a result.

Given that multiple sub-elements of v may be associated with w, we use the following formula for the overall score of v given w

$$Score(v, w) = max_{u \in Desc(v) \cup v} PS(v, w, u) \tag{7.3}$$

Other monotonic aggregation functions are also possible. The score of a result element v for Q is

$$Score(v, Q) = \sum_{w \in Q} Score(v, w) \tag{7.4}$$

Again other monotonic aggregation functions are possible.

Association degree of node to keyword: The association degree $NS(v, w)$ of node $v \in T$, $T \in D$ with respect to a keyword w, given documents collection D and an ontological systems collection O is a combination of its IR score with respect to w and its ontological association to w.

$$NS(v, w) = max \left(\begin{array}{l} IRS(v.text, w), \\ OS_{v.onto.system}(CN(v.onto), w) \end{array} \right) \tag{7.5}$$

where $IRS(d, w)$ is the IR score of a document d given keyword w within the collection D. D is an implicit input to $IRS(\cdot)$ since popular IR functions [Sal89, RW94a, Sin01] use the document frequency (*df*) which is computed over D. We view each XML element as a document to apply the IR function. In our experiments we use the BM25 [RW94a] function.

$OS_{v.onto.system}(u, w)$ is the association degree (*OntoScore*) of a node (concept) $u \in O_i$, where O_i is specified by $v.onto.system$, to keyword w, and is computed by exploiting the relationships in O_i, as explained in detail in Section 7.3.

$CN(v.onto)$ returns the concept node with code $v.onto.concept$ in the ontological system specified by $v.onto.system$. For instance, consider the document of Figure 9.1 shown in the Appendix, and the ontological system of Figure 6.2 in Section 6.2. $CN(v.onto)$ for the code element v of Line 39 in Figure 9.1 will return the concept node *"Asthma"* identified with the code 195967001 in Figure 6.2 in Section 6.2. $IRS(\cdot)$ and $OS(\cdot)$ are normalized to $[0, 1]$.

The intuition of (7.5) is that a node v may be associated with a keyword w either through its textual description $v.text$ or through its ontological reference $v.onto$. We then pick the strongest one. The $OS(\cdot)$ term of a non-code node is 0. Again, alternative monotonic aggregation functions are possible.

For instance, for the keyword *w="Asthma"* assuming node v of Line 39 in Figure 9.1 has $IRS(v.text, w) = 0.3$ and its related SNOMED node u has $OS_{SNOMED}(u, w) = 0.5$, its $NS(v, w)$ would be 0.5.

7.3 Semantic Relevance of Ontological Concepts to Keywords

A key component of XOntoRank is the derivation of semantic relevance of a concept v in the ontology to a query keyword w. Since nodes in an XML document may refer to concepts in the ontology, this derivation essentially quantifies the semantic relevance of an XML element to a query keyword based on terminological definitions in the ontology.

The Semantic Web community has developed various mechanisms to determine semantic similarity of concepts in an ontology (see Section 7.7 for a description of Related Work). However, most existing measures do not use relationship information between concepts in a general manner. The main advantage of ontologies like SNOMED over simpler taxonomies is that they describe various kinds of relationships between concepts, which can be used to calculate relevance measures.

We view the ontology as a graph, where the nodes in the graph represent concepts, and edges represent relationships between concepts. Our approach for calculating the semantic relevance of a concept to a query keyword is inspired by the idea of authority flow. Initially, each concept in the ontology is granted a certain authority based on how strongly it is related to w, as measured by its IR score. Authority then flows from these concepts to other concepts in the ontology based on certain rules. Note that the authority flow occurs in a recursive fashion and hence, it can affect descendants and not only direct children of the involved elements.

In this section, we examine various strategies for directing the flow of authority, based on different views of the ontology. For simplicity of presentation we consider a single ontology O_0 and omit the O_0 subscript at $OS()$. We use the overloaded function $OS(v, w, x)$ to represent the relevance of concept v to keyword w due to the occurrence of w in another node x in the ontology. It is:

$$OS(v, w) = max_{x \in O_0}(OS(v, w, x)) \tag{7.6}$$

Other monotonic aggregation functions are possible.

7.3.1 View Ontology as Undirected, Unlabeled Graph

This strategy treats the ontology as an undirected graph, with no distinction among the different kinds of relationships between concepts. Based on this view, we define $OS(v, w, x)$ as:

$$OS(v, w, x) = IRS(x, w) \cdot decay^{l} \tag{7.7}$$

where $l = distance(v, x)$ and $0 \leq decay \leq 1$.

7.3.2 View Ontology as Taxonomy

This strategy only considers the taxonomic portion of the ontology, i.e. we only consider ***is-a*** links between concepts for calculating $OntoScore$. The ***is-a*** links form a Directed Acyclic Graph ***(DAG)***, since cycles are not permitted based on subclass relationships. $OS(v, w, x)$ is computed recursively using (7.6) and the following two cases:

i ***x* is a superclass of v,** i.e., there is a path from v to x in the DAG formed by the *is-a* links. In this case,

$$OS(v, w, x) = IRS(x, w)$$

The intuition behind this definition is that since x is a superclass of v, any query for x is completely and logically satisfied by v. For example, let v be *"Asthma"*, w be *"Bronchus"* and x be *"Disorder of Bronchus"* (*"DOB"*) in the ontology fragment of Figure 6.2. It is *OS("Asthma", "Bronchus", "DOB") = IRS("DOB", "Bronchus")*. An extreme case of this rule is when x is the same as v. In this case, $OS(v, w, v) = IRS(v, w)$.

ii x **is a direct subclass of v,** i.e. there is an *is-a* link from x to v. In this case,

$$OS(v, w, x) = IRS(x, w) \cdot (1/n)$$

where n is the number of subclasses of v. The intuition behind this definition is that since x is a subclass of v, any query for x is partially satisfied by v. Our heuristic for calculating the extent of the partial satisfaction is based on the number of subclasses of v, similarly to the authority flow distribution in [BHP04]. For example, let v be *"Disorder of Bronchus"*, w be *"Asthma"* and x be *"Asthma"* in Figure 6.2. In the actual ontology, the concept *"Asthma"* has 26 direct subclasses. Hence, in this case, *OS("Disorder of Bronchus", "Asthma", "Asthma") = IRS("Asthma", "Asthma") *(1/26)*.

7.3.3 Including the Relationships between Concepts

To handle different kinds of relationships, we interpret concepts and relationships in SNOMED using description logics [Baa03]. Many biomedical ontologies, including SNOMED, belong to a category of Descriptions Logics called $\mathcal{EL}^+$ [BLS06]. Concepts in this logic are defined as follows:

$$C ::= A|T|C \sqcap D|\exists r.C \tag{7.8}$$

where A ranges over atomic concept names

T is the top concept

r ranges over relationship names

C, D are concept names

$\sqcap$ is the concept intersection operator

The $\exists r.C$ construct is an existential quantification operator that declares the existence of a relationship (or role) to a concept C. We can also view $\exists r.C$ as a concept where every instance of the concept is related by role r to an instance of a concept C. We call such a concept an *existential role restriction*, since it describes a constraint or restriction on the values of a relationship. (7.8) describes the different ways in which a concept can be defined in the $\mathcal{EL}^+$ logic. The $\mathcal{EL}^+$ logic also defines subclass (or concept inclusion) relationships between concepts as $C \sqsubseteq D$.

Some examples of $\mathcal{EL}^+$ expressions from Figure 6.2 are:

$$\begin{aligned}\text{Disorder of Thorax} &\sqsubseteq \text{Finding of Region of Thorax}\\ \text{Asthma Attack} &\sqsubseteq \text{Asthma}\\ &\sqcap \exists\text{Finding-site-of.Bronchial Structure}\end{aligned}$$

Consider the last statement, which says that *"Asthma Attack"* is a concept that is a subclass of Asthma and that has a *finding-site-of* relationship to the *"Bronchial Structure"* concept. In other words, any instance of *"Asthma Attack"* (e.g. the *"Asthma Attack suffered by"* a specific patient) is also an instance of *"Asthma"* and is found in some instance of *"Bronchial Structure"*.

This description logic view allows us to describe every concept as a subclass of a set of atomic concepts or existential role restrictions. Hence, we can reduce a graph with different kinds of relationships into one that has only subclass or *is-a* relationships.

For example, consider an ontology graph fragment depicted in Figure 7.1. A description logic view of this ontology would appear as shown in Figure 7.2. The dotted links between concepts represent *is-a* links, meant to indicate the relationship between a concept X and a $\exists r.X$ for any role r.

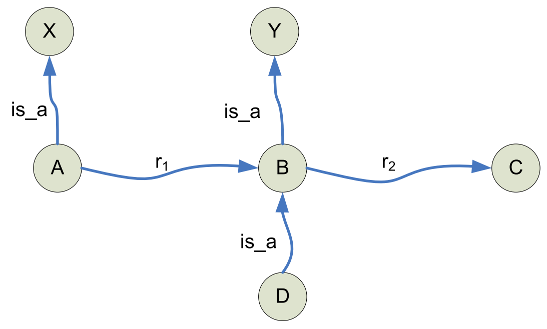


Figure 7.1: Sample Ontology Fragment.

We now calculate $OS(v, w, x)$ in this logically transformed ontology graph using an extension of the strategy of Section 7.3.2. In particular, if there is a "dotted link" between x and v, i.e. one of x or v is of the form C, and the other is of the form $\exists r.C$, then,

$$OS(v, x, w) = OS(x, w) \cdot \alpha \tag{7.9}$$

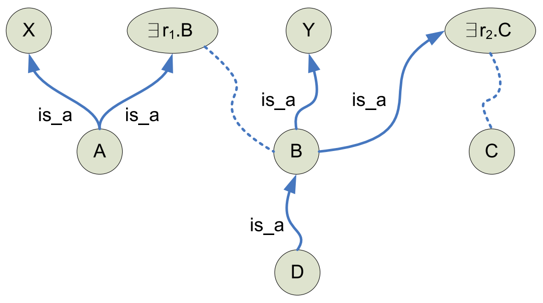


Figure 7.2: Ontology's Description Logic View.

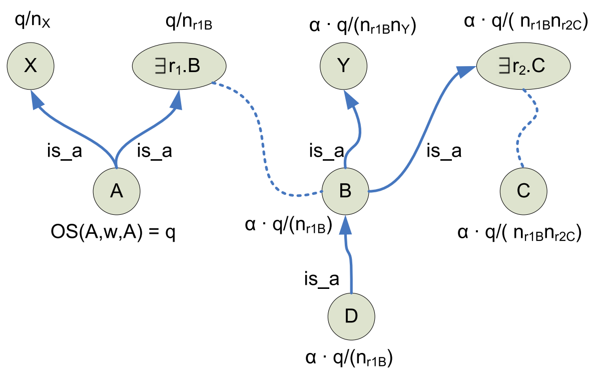


Figure 7.3: OntoScore Propagation. n_i is the number of subclasses of node i.

Here, α represents the decay in semantic relevance when traversing a dotted link between a concept C and a role restriction $\exists r.C$.

As an example, assuming that $OS(A, w, A) = q$, then the *OntoScore* would propagate as shown in Figure 7.3 to different nodes in the ontology.

We provide a syntactic name to the concepts corresponding to existential relationship restrictions so as to allow calculating $IRS(x, w)$ when x is a role restriction concept of the form $\exists r.C$. The syntactic name in our implementation is *"Exists"+r+C*. For example, the relationship *"finding site of"* between *"Asthma Attack"* and *"Bronchial Structure"* in Figure 6.2 gives rise to the new existential role restriction named *"Exists finding site of Bronchial Structure"*.

7.4 Architecture and System Overview

In this section we present the architecture and overview of the XOntoRank system.

7.4.1 XOntoRank Architecture

Figure 7.4 shows the architecture of XOntoRank, which is divided into two stages. The preprocessing phase consists of the Index Creation Module, which takes as input the corpus of XML-formatted EMR documents to be indexed (CDA in our experiments), the ontological system(s) referenced in the EMR documents and the set of all keywords (the vocabulary) to be indexed.

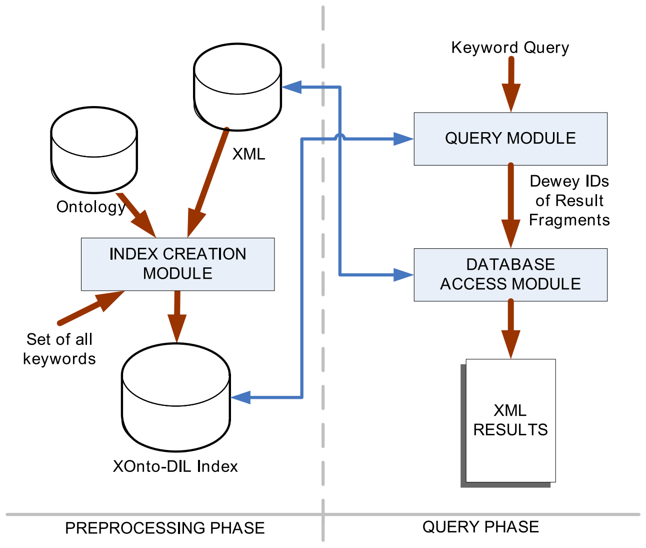


Figure 7.4: XOntoRank Architecture.

The Index Creation Module generates the *XOntoRank Dewey Inverted Lists (XOnto-DILs)* which are inspired from the Dewey Inverted Lists of XRANK [GSBS03]. XRANK is based on *ElemRank*, a variation of the PageRank algorithm that exploits the structure and containment edges of XML documents. The key difference is that instead of $ElemRank(v)$ we store $NS(v, w)$, that is, the relevance score of node v with respect to keyword w given the XML documents and the ontological systems, defined in (7.5). *ElemRank* could be incorporated in $NS(v, w)$ but our CDA documents have no ID-IDREF edges and hence *ElemRank* would make no difference.

For example, Figure 7.5 shows the Dewey ID's generated for a subset of the document of Figure 9.1. We have truncated the prefix in the Dewey ID's for space constraints. Figure 7.6 shows a fragment of the *XOnto-DIL* for the same document. Note that the first component of each Dewey ID is the document ID. The process to build *XOnto-DIL*s is described in detail in Section 7.5.2.

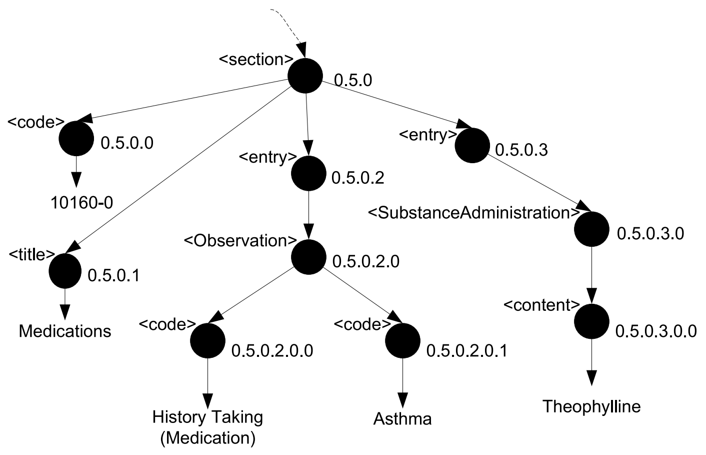
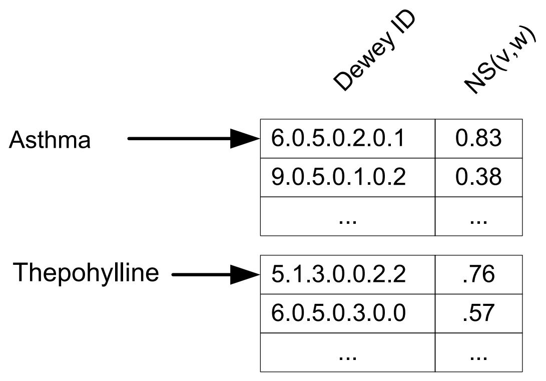


Figure 7.5: Dewey IDs for CDA Document.



	Dewey ID	NS(v,w)
Asthma	6.0.5.0.2.0.1	0.83
	9.0.5.0.1.0.2	0.38
	...	...
Thepohylline	5.1.3.0.0.2.2	.76
	6.0.5.0.3.0.0	.57
	...	...

Figure 7.6: Dewey Inverted List for CDA Document.

During the query phase, the Query Module inputs the user keyword query and executes XRANK's DIL algorithm using the XOnto-DILs generated in the pre-processing phase. The Database Access Module then obtains the appropriate XML fragments addressed by the resulting Dewey ID's.

7.4.2 Building the XOnto-DILs

In this section we describe how the *XOnto-DIL*s are computed for the various semantics described in Section 7.3. We compute *XOnto-DIL*s for all words in the Vocabulary, defined as the union of words in the ontological systems $O_1, \ldots, O_s$ and in documents in D. As above, we assume there is a single ontological system O_0. *XOnto-DIL*s are computed in three stages:

Full-text Indexing: First, we build a full-text index of the CDA documents and the ontology. This phase is common to all the algorithms, and computes the TF-IDF score.

OntoScore Computation Stage: Second, we build an *OntoScore Hash Map* M, that stores the $OS(v, w)$ for every pair (v, w) of concept node v and keyword w with $OS(v, w) > threshold$, where *threshold* is a predefined value used to improve the efficiency of building M. We chose a *threshold* that could give us a balance of space and quality. The details of computing M, as well as the criteria to choose *threshold* are presented in Section 7.5.

DIL Creation: Finally, we compute the XOnto-DILs for the documents in D. The $NS(v, w)$ for each pair (v, w) of node $v \in T_i$, $T_i \in D$, $w \in Vocabulary$ is computed by (7.5), where $OS(CN(v.onto), w)$ is retrieved from Hash Map M. We show how M is computed in the next section.

7.5 OntoScore Computation Algorithms

In the next sections we show how the Hash Map M is computed during the OntoScore stage for each of the OntoScore computation methods described in Section 7.3.

Listing 7.2: Compute OntoScore Hash Map.

```
1  procedure ComputeOntoScore(Vocabulary $V$, SNOMED Ontology Graph $O$)
2  for each keyword $w$ in $V$
3  begin
4    /* Find all concept nodes in $O$ that contain $w$
5    $S \leftarrow$ getRootSet($w$, $O$)
6    for each concept $s \in S$
7    begin
8      do BFS from $s$
9      for each accessed concept node $v$
10     begin
11       Compute $OS(v,w)$ /* By Eq. 7.7 */
12       /* If expanding $u \rightarrow v, OS(v,w) = OS(u,w) \cdot decay$ */
13       if $M.get(v,w) < OS(v,w)$
14         $M.put((v,w), OS(v,w))$
15       else
16         Stop BFS expansion for $v$
17       end if
18     end
19   end
20 end
```

7.5.1 Ontology as Undirected Graph

If a node $v \in O_i$ can be reached from multiple concept nodes $u_1, \ldots, u_x$, then we assign to u the maximum score that any of $u_1, \ldots, u_x$ would assign. Again other aggregation functions are possible.

$$OS(v,w) = max_{i=1\ldots x}(OS(v,w,u_i)) \tag{7.10}$$

The algorithm to compute the Hash Map M in the *OntoScore* phase is depicted in Listing 7.2.

An inefficiency of Listing 7.2 is that it does breadth-first-search (BFS) starting from all nodes that contain keyword w (Line 4). This can potentially lead to traversing the same node multiple times, once for each BFS instance. This can be avoided using the following observation:

Observation 1: *If multiple BFS instances arrive at a node, then we only need to propagate one value, which corresponds to the aggregate function, that is, we merge the met BFS expansions into* *one with the aggregate node score.*

The reason is that the score propagates by multiplying by decay for each level. Hence, if v has score $f(OS_i, OS_j)$ where $f(\cdot)$ is the combining function (max in (7.10)), a node u with distance l from v will have score $f(OS_i, OS_j) \cdot decay^l$. If we would ignore this observation and do the BFS expansions independently, u would get score $f(OS_i \cdot decay^l, OS_j \cdot decay^l)$. The two quantities are equal for any reasonable combining function $f(\cdot)$ like max, sum, and $product$.

The above observation is implemented by doing the following changes to Listing 7.2: We replace Line 4 by the following:

```
4   do BFS in parallel from $s$
```

and insert the following lines after Line 6:

```
7   if $v$ already has an $OS$ score then
8     Stop expanding $v$ for expansion instance that produced
          the smallest $OS(v, w)$
```

Note that to do BFS in parallel we insert all nodes in S in the BFS queue and then do BFS as usual. To halt the expansion of a node v (Line 6.2 in the correction above) that has already been processed and its adjacent nodes C have already been inserted in the queue, we maintain pointers from v to C in the queue, and remove from the queue the nodes in C when v's expansion is halted.

7.5.2 Ontology as Taxonomy

As mentioned in Section 7.3.2, we restrict the links used to compute *OntoScore*, by only considering the *is-a* and *inverse-is-a* edges in SNOMED. Hence, the first modification is to change the loop in Line 3 of Listing 7.2 to restrict the BFS to only follow these two types of relationships, capturing only the taxonomic portion of the ontology.

We also modify the way in which $OS(v, w)$ is computed (Line 5 of Listing 7.2), replacing the formula in (7.7) by the cases exposed in Section 7.3.2. In particular, if we expand from node u with $OntoScoreOS(u, w)$ to node v, then:

- if $u \xrightarrow{is-a} v$ then $OS(v, w) = \frac{OS(u,w)}{InDegree_{is-a}(v)}$
- if $u \xleftarrow{is-a} v$ then $OS(v, w) = OS(u, w)$

where $InDegree_r(v)$ is the number of incoming relationship edges of type r.

The rest of the algorithm stays as specified in Listing 7.2, using the same threshold constraints and the same optimization described in *Observation 1*.

7.5.3 Ontology as Collection of Relationships

In this case, as mentioned in Section 7.3.3, all relationship edges are considered. We enumerate below how the expanded nodes are assigned *OntoScores* without having to physically create the ontological graph with the existential role restrictions described in Section 7.3.3. The assigned *OntoScores* are equal to the ones computed by building the ontological graph described in Section 7.3.3.

Hence, the BFS expansion is the same as in Section 7.4.1. The *OntoScore* computation of Line 5 is changed as follows, to reflect the approach described in Section 7.3.3. If we expand from node u with *OntoScore* $OS(u, w)$ to node v, then:

- if $u \xrightarrow{is\_a} v$ then $OS(v, w) = \frac{OS(u,w)}{InDegree_{is\_a}(v)}$
- if $u \xleftarrow{is\_a} v$ then $OS(v, w) = OS(u, w)$
- if $u \xrightarrow{r} v, r \neq is\_a$ then $OS(v, w) = a \cdot \frac{OS(u,w)}{InDegree_r(v)}$
- if $u \xleftarrow{r} v, r \neq is\_a$ then $OS(v, w) = a \cdot OS(u, w)$

Note that the denominator $InDegree_r(v)$ is the in-degree of the existential role restriction $\exists r.v$.

7.6 Experiments

In this section we experimentally evaluate the XOntoRank system and show the feasibility of both the Preprocessing and Query phases. The experiments were performed on a Pentium 4, 2.8 GHz

PC with 1GB RAM. XOntoRank was implemented in Java JDK 5.0, using DOM for XML parsing and Microsoft SQL Server 2000 for the persistent storage of indexes. To access and navigate SNOMED CT, which takes multiple GBs of disk space, we used the API provided by the National Library of Medicine (NLM) Unified Medical Language System (UMLS) [NLM08]. This API provides the necessary methods to query the ontology and dictionary and obtain the concept code and display name for a particular string. We used this API as a black box in both the preliminary CDA document generation and the Index Creation Module of XOntoRank.

Table 7.1: Number of results marked as relevant for each query. User marks up to 5 results.

	Query	**XRANK**	**Graph**	**Taxonomy**	**Relations**
q_1	“cardiac” “arrest”	5	5	5	5
q_2	“cardiac” “coarctation”	5	5	5	5
q_3	“neonatal” “cyanosis”	3	3	0	3
q_4	“carbapenem” “ibuprofen”	0	3	0	3
q_5	“supraventricular arrhythmia” “pericardial effusion”	0	0	1	0
q_6	“regurgitant flow” “amiodarone”	0	1	1	2
q_7	“supraventricular arrhythmia” “acetaminophen”	0	0	0	0
	AVERAGE	1.875	2.429	1.714	2.571

In Section 7.6.1 we quantify the differences in the ranking for the alternative OntoScore computation techniques of Section 7.3. We also present results of a user survey that we performed with the aid of a medical doctor and researcher. In Section 7.6.2 we measure the performance of the XOntoRank system in terms of index creation and query execution times. Some screenshots of the XOntoRank system are available at the project homepage [Flo08]. The system was not made available to the public due to patient record privacy concerns.

CDA Documents Generation: We developed a program to convert automatically the relational anonymized EMR database of the Cardiac Division of a local hospital into a set of XML CDA documents. Each CDA document represents the medical record of a single patient conglomerating all her hospitalization entries. 3 492 such documents were created, each being on average 47KB with 1 133 XML elements. Ontological references were inserted for every XML node whose value matched one of the concepts in SNOMED. This resulted in 2 454 CDA documents with ontological references to SNOMED with an average of 151 references per document.

7.6.1 Quality Results

We performed two quality experiments. The first one compares the distances between the result lists of the proposed search approaches for a real query workload, and the second one is a proof-of-concept user survey which compares the user satisfaction for these approaches. The four approaches –baseline plus the three described in Section 7.3– are denoted as *XRANK* (baseline, no use of ontology), *Graph* (Section 7.3.1), *Taxonomy* (Section 7.3.2), and *Relationships* (Section 7.3.3).

Distance between Top-k lists: We performed a series of two-keyword queries obtained from domain expert collaborators. The second column of Table 7.1 shows a sample of these queries. Note that some keywords are phrases enclosed in quotes. We use the top-*k* Kendall Tau [FKS03] measure to determine the distance between the lists and hence test the effects of each individual algorithm. Table 7.2 reports the Kendall Tau values for $k = 20$ and penalty parameter $p = 0.5$ (see [FKS03] for definition of p), normalized over 20 queries. We observe the large distance between the result of *Graph* and the *Relationships* algorithm; this was expected since the expansion on the ontology graph achieved by the *Graph* algorithm is less restricted than the *Relationships* algorithm, which extends the *Taxonomy* expansion. For this reason, the distance between *Taxonomy* and *Relationships* lists is small.

Table 7.2: Normalized Kendall Tau values for four approaches.

	XRANK	*Graph*	*Taxonomy*	*Relationships*
XRANK	0.000	0.171	0.101	0.209
Graph	0.171	0.000	0.116	1.000
Taxonomy	0.101	0.116	0.000	0.171
Relationships	0.209	1.000	0.171	0.000

Quality Survey: We conducted a survey to determine the quality of each of the four algorithms we presented. Given the specialized nature of our medical records dataset, which come from a children's cardiac clinic, it is hard to find many users to properly evaluate the results. Hence, we chose to only report, as a proof of concept, the results of a survey on a single domain expert–medical doctor and researcher knowledgeable in this area–instead of involving non-expert users who could degrade the reliability of the results.

The results of the survey are shown in Table 7.1. For each query, we presented to the user the union of the top-5 results from each of the four algorithms. The user was asked to select up to 5 results that he found relevant to the query. For this experiment, we set *decay* to 0.5, *threshold* to 0.1 and α to 0.5.

For queries q_1 and q_2, the top-5 results obtained by *XRANK* are also the top-5 results for the ontology-enabled algorithms, because the query keywords appear frequently in the CDA documents. For q_3, *XRANK* only generated three results –all of which were marked as relevant–, but only one of these appear in the top-5 list of the other three algorithms. For the remaining queries, *XRANK* does not produce any results, since there is no CDA document with direct occurrences of both keywords (or phrases). In contrast, the ontology-enabled algorithms find relevant results to the queries by mapping the keyword's concept to other concepts present in the documents. For q_4, both *Graph* and *Relationships* algorithms produce the same results by expanding through non-taxonomical edges in the SNOMED ontology.

For q_5, only the *Taxonomy* algorithm produced a result that was considered "relevant" by the domain expert. This result did not reach the top-5 of *Graph* and *Relationships* algorithms, because the expansion through non-taxonomical concepts produced more compact results –single XML elements that mapped a concept to both query keywords– with higher score, but those were not considered relevant by the domain expert.

For q_6, the *Relationships* algorithm produces better results, because it combines the results of both the *Graph* and *Taxonomy* algorithms; the expansion over the ontology for the *Graph* algorithm decayed before it could reach the taxonomical result found by the *Taxonomy* and *Relationships* algorithms.

Note that in some cases, the semantic knowledge represented by the ontology might not be sufficient to provide high quality Information Retrieval over EMR's. For instance, consider query q_7 =[*"supraventricular arrhythmia" "acetaminophen"*]. The scores of zero for the ontology-assisted algorithms in Table 7.1 are due to the following reason: All the results of these algorithms map the concept *"acetaminophen"* to the concept *"aspirin"*. In the context of *pain control*, these two concepts are indeed related, because they both provide relief of pain. But in this specific case, the keyword *"supraventricular arrhythmia"* implies that the target context of this query is not *pain control* but *cardiology*, and in this context, however, these drugs are generally unrelated. *"Aspirin"* has cardiac benefits that are not seen with *"acetaminophen"*, due to the differing properties of the two drugs.

The findings of Table 7.1 are summarized as follows. The quality of *Relationships* and *Graph* is generally superior to the baseline *XRANK* algorithm, which means that when the keywords are not present in a document, the ontology-enhanced algorithms are capable of finding "good" results to satisfy the given queries. The *Taxonomy* algorithm can be slightly worse than *XRANK*, since the former could return results where a query keyword is matched to a far ancestor concept, because *Taxonomy* does not penalize the ontology expansion when following *is-a* (parent) edges.

7.6.2 Performance Results

Pre-processing phase: Building XOnto-DIL lists for all keywords in the SNOMED ontology was not feasible given that they are in the order of millions, the keywords vocabulary cannot be extracted from the provided SNOMED API, and the API is slow given that it is IO-intensive (note that SNOMED is a multi-gigabyte ontology). Note that there is a method to get all occurrences of a specific keyword, but there is no vocabulary of all keywords in the database. Hence, we indexed a subset of this universe of keywords which let us execute a large number of queries and estimate reliable projections of index execution time. In particular we built XOnto-DIL lists for all the keywords in the CDA documents and for all keywords contained in a concept, up to 2 relationships away from a concept referenced in a CDA document (more than 400 unique concepts are referenced in our CDA collection). The above rules translated to the indexing of more than

40 000 keywords directly present in the documents and more that 100 000 concepts from the SNOMED ontology. To navigate SNOMED efficiently, we loaded the appropriate fragment in main memory, thus reducing the access to SNOMED flat files. However, the SNOMED navigation was still too slow. In the future, we plan to work on more efficient ways to navigate the ontology to build the XOnto-DIL lists, as discussed in Section 9. We set *decay* to 0.5, *threshold* to 0.1 and α to 0.5.

Table 7.3 presents the average creation time, average number of postings (rows in Figure 7.6) and size of a XOnto-DIL list of a keyword for each of the four approaches. For the average creation time, we exclude the time taken to navigate the SNOMED ontology, since it can take up to several minutes for frequent keywords, given the current implementation of the SNOMED API.

We observe that the average creation time for *Taxonomy* is much larger than *Graph*. This is due to the fact that the expansion in *Graph* decays continuously, whereas the expansion for *Taxonomy* decays quickly only for descendants, but may expand indefinitely for parent relationships. We also see how the *Graph* and both *Relationships* approaches generate the largest number of XOnto-DIL entries, given the fact that the navigation does not decay for the one direction of *is-a* edges. We observe a high difference between the number of postings for the *Taxonomy* approach compared to the *Relationships* algorithm, giving evidence of the large number of concepts mapped through the ontology graph. Note that the size of the XOnto-DIL entries can be reduced by appropriately adjusting the *threshold* and/or *decay* parameters.

Table 7.3: Average Size for XOnto-DIL Entries.

Algorithm	*Per Keyword*		
	Avg. Creation Time (ms)	*Postings*	*Size (KB)*
XRANK	1.0	1 435.7	39.3
Graph	4 143.5	20 906.7	571.7
Taxonomy	10 743.5	5 511.9	150.7
Relationships	13 485.3	46 979.5	1 284.6

Query Phase: Figure 7.7 presents the average execution times for queries with varying number of keywords, for $k = 10$. The time for *Relationships* algorithm is higher due to the larger number of nodes in the XML document that are ontologically related to the query keywords.

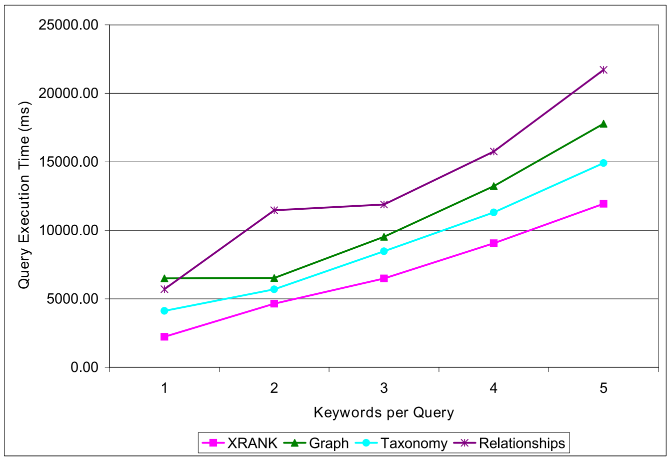


Figure 7.7: Average Execution Time for Keyword Queries with Varying Number of Keywords.

7.7 Related Work on Leveraging Ontologies for Information Retrieval

Various query expansion strategies (e.g. [XC96]) have been proposed for general as well as biological documents search. For instance, the QEEF framework [WR05] uses the UMLS ontology to suggest additional terms. [The03, STW03, STW05], assign weights on the ontology edges by comparing the distributions of the contents of the two nodes and of their combination on a very large dataset like the Web. This approach, which complements our work, is too time-consuming for large ontologies like SNOMED. The ontological associations are exploited by expanding the XXL query. It differs from our approach in which XXL considers symmetric associations between ontology concepts, whereas we use the authority flow model. [KK05, KKJ06] expand the query by matching the ontology to the document DTD. All the above techniques are proposed for structured XML queries. For our case of keyword queries, query expansion is not appropriate, since it leads to non-minimal results (see [HP02] for a definition of a minimal keyword search result) — the same concept appears multiple times in a result.

In Information Retrieval, two approaches have addressed the problem of computing similarity between two concepts. Initially, statistical correlations between terms were exploited [Les69]. With the conception of ontologies and semantic networks like WordNet [Fel98], a graph-oriented approach was adopted, focusing on the number, depth and direction of the edges between two concepts [RMBB89]. A more recent approach has combined these two techniques [Lin98, Res99] by taking into account the graph structure and statistics.

In the Semantic Web, various approaches have been suggested to measure semantic similarity between different artifacts. Most similarity measures such as [LH03, KC06] focus only on subsumption relations (i.e. hierarchical *"is-a"* links in an ontology). Maguitman et al. [MMRV05] propose an information theoretic measure of similarity that also considers non-hierarchical links. However, their approach requires the presence of a large number of instances to determine the similarity between concepts. In the medical domain, most ontologies, including SNOMED, only describe concepts and not instances. Hence, their approach cannot be used. The notion of authority flows is also similar to the spreading activation scheme that is used in information retrieval [Cre97] and web mining [GVD05]. A novel aspect of our approach is the use of strategies based on description logics and the spreading of activation from the ontology into the XML documents.

7.8 Conclusions and Future Work

We have introduced the problem of ontology-aware keyword search on XML-based EMR documents, which contain references to clinical ontological concepts. We defined semantics for this problem, where the ontological references, as well as the relationships within the ontology are used in creating and ranking the query results. Alternative views of the ontology were considered. We created efficient algorithms, building on previous work, to generate the top-k query results. The algorithms were evaluated experimentally, showing that the precision and recall of our algorithm is better than the baseline algorithm.

A critical future direction is the optimization of the index creation process. Our current index creation approach relies on the API and data provided by [SNO08], which are based on flat files. Implementing approximation and early pruning techniques, as well as in-memory representations of the ontology graphs, may prove useful in scaling to larger ontologies and datasets.

CHAPTER 8

COMPARING TOP-K XML LISTS

8.1 Introduction

Systems that produce ranked lists of results are abundant. For instance, Web search engines return ranked lists of Web pages. To compare the lists produced by different systems, Fagin et al. [FKM$^+$04, FKS03] present distance measures for top-k lists that extend the traditional distance measures for permutations of objects, like Kendall tau [FKM$^+$04] and Spearman's Footrule [FKM$^+$04].

In addition to ranking whole objects (e.g., Web pages), there is an increasing number of systems, including XRANK [GSBS03], XSEarch [CMKS03], XKeyword [HPB03], XXL [TW02a, TW02b], XIRQL [FG01], that provide keyword search on XML or other semi-structured data, and produce ranked lists of XML sub-trees. In addition, XML lists distance measures can also be applied to rank-aware extensions [FG01] of XPath and XQuery. Furthermore, these measures are needed for XML lists aggregation, where the results from several XML search engines can be aggregated to find the best top-k list for the given lists. [DKNS01] presents the Web page aggregation problem. Clearly, there is a need to have measures to compare the results of such systems among each other or against the user's ideal list of results.

Unfortunately, previous distance measures are not suitable for ranked lists of sub-trees since they do not account for the possible overlap between the returned sub-trees. That is, two sub-trees differing by a single node would be considered separate objects. For instance, Figure 8.1 shows two top-3 lists of sub-trees produced by two imaginary XML keyword proximity search algorithms. Trees Ta_2 and Tb_3 only differ by a single node but this is ignored by object-level distance measures.

In this chapter, we present the first distance measures for ranked lists of sub-trees, and show under what conditions these measures are metrics. In particular, the distance measures consist of two components: the tree similarity component and the position distance component. The former captures the similarity between the structures of the returned sub-trees, while the latter captures the distance of the sub-trees in the two lists, similarly to previous object-level distance measures [FKM$^+$04, FKS03].

Intuitively, our distance measures work in two phases. In the first phase, they find the optimal (closest) mapping between the two top-k lists of sub-trees, where the distance between a pair of sub-trees is computed using one of the approaches proposed in previous works, including tree edit distance [Bil03, Bil05, LCS$^+$04, NJ02], tree alignment distance [Bil03], Fourier transform-based similarity [FMPP02, FMMP05], entropy-based similarity [Hel07], tag similarity [But04], and path shingle similarity [But04]. The cost of the optimal mapping between the two lists of sub-trees represents the tree similarity component.

Next, we compute the position distance component given the optimal mapping, using one of the previously proposed techniques on measuring the distance between top-k (partial) lists [FKM$^+$04, FKS03].

In the rest of the chapter we focus on XML trees; however the exact same ideas can be applied to any type of tree representations. We make the following contributions:

1. Present the first suite of distance measures for ranked lists of sub-trees. Three variants are presented. The XML Lists Similarity Distance based on Total Mapping (XLS) where all sub-trees from the first list are mapped to sub-trees in the second, XML Lists Similarity Distance based on *Total Mapping* with position component (*XLS-P*) which includes a position component in addition to the XML similarity component and the XML Lists Similarity Distance based on *Partial Mapping with position component* (*XLS-PP*) where only adequately similar sub-trees are matched to each other.

2. Prove under what conditions these measures are metrics. As we show, the trickiest requirement is the satisfaction of the triangle inequality.

3. Present efficient algorithms to compute *XLS*, *XLS-P* and *XLS-PP* for two lists of XML sub-trees.

4. We conducted a study to compare three popular XML keyword proximity search systems: XRANK [GSBS03],

   XSEarch [CMKS03] and XKeyword [HPB03]. We implemented all three systems and

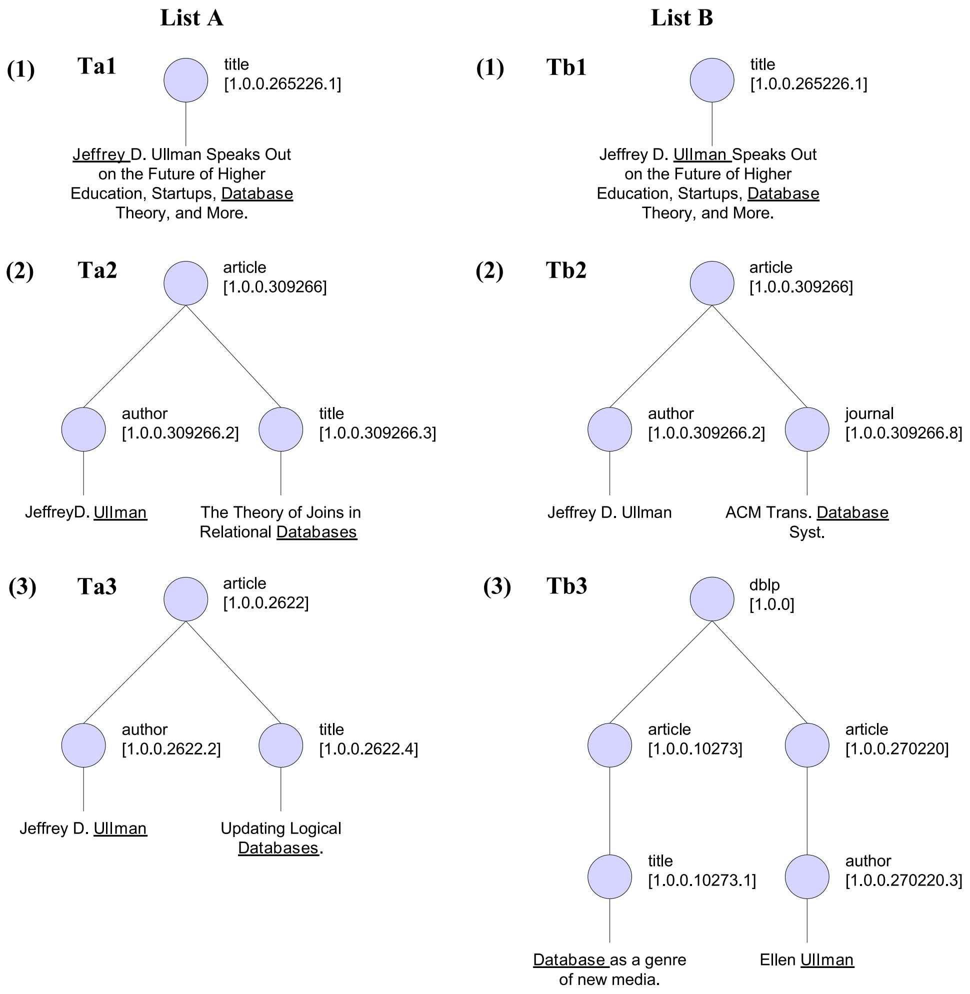


Figure 8.1: Top-3 trees for query *“Ullman Database”*.

report on the *XLS*, *XLS-P* and *XLS-PP* distances of their results for various datasets and queries.

The rest of this chapter is organized as follows: Section 8.2 presents the background. Section 8.3 presents the distance measures for lists of XML trees. Section 8.4 briefly describes our distance measures for various tree similarity measures described in Section 8.2.1. Section 8.5 describes the normalization issues. Section 8.6 presents algorithms for computing the proposed XML list distance measures. Section 8.8 presents our experimental evaluation and Section 8.7 presents the related work.

8.2 Background

In this section we briefly discuss various tree similarity measures (Section 8.2.1). We then discuss some of the popular distance measures for lists of objects (Section 8.2.2) and the conditions that a measure must satisfy to be considered a metric (Section 8.2.3).

8.2.1 Tree Similarity Measures

In this section we briefly present state-of-art techniques for measuring similarity between trees proposed in the literature. Any of these similarity measures can be used in our framework. However, only the measures that are metrics will lead to a distance metric for XML lists, as shown in Section 8.3.

General Tree Similarity Measures: Several techniques have been proposed in the literature for measuring the similarity between general trees. Tree edit distance [Bil05, Tai79, YKT05, ZS89] measures the minimum number of node insertions, deletions, and updates required to convert one tree into another.

Tree alignment distance [Bil05, JWZ94] is a special case of the tree editing problem, in which trees become isomorphic when labels are ignored.

XML-Specific Tree Similarity Measures: Various techniques for measuring the structural similarity between XML trees have been proposed. All of these measures were used to cluster XML documents based on structure. Jagadish et al. [NJ02] introduced a structural similarity distance based on tree edit distance, by adding insert-tree, delete-tree operations in order to develop an edit distance metric that is more indicative of the structural similarity between XML trees. Flesca et al. [FMMP05] propose a Fourier transform technique to compute similarity. Buttler [But04] presents a similarity metric based on path-shingles in which the structural information is extracted from the documents using the Full Paths. Entropy-based similarity [Hel07] is a novel technique used to compute the structural similarity of semi-structured documents based on entropy. Tag similarity is perhaps the simplest metric for structural similarity, as it only measures how closely the set of tags match between two pages. [WN05] discusses a method to identify duplicate entities in a XML document which could be used to enhance the tree mapping step in our distance metrics.

8.2.2 Distance Measures for Permutations

In this section we present some of the most popular and widely used measures for the distance between complete lists of objects (permutations). We review Spearman's footrule and Kendall tau distance measures [Dia88, FKS03, KG90]. Spearman's footrule metric is the $L1$ distance between two permutations. Formally, it is defined by

$$F(\sigma_1, \sigma_2) = \sum_{i=1}^{k} |\sigma_1(i) - \sigma_2(i)|$$

where σ_1 and σ_2 are the two permutations of length k, and $\sigma_1(i)$ denotes the ith element in σ_1.

Kendall tau metric between permutations is defined as follows: For each pair $i, j \in P$ of distinct members, if i and j are in the same order in σ_1 and σ_2, then let $\overline{K}_{i,j}(\sigma_1, \sigma_2) = 0$; else $\overline{K}_{i,j}(\sigma_1, \sigma_2) = 1$. Kendall tau is

$$K(\sigma_1, \sigma_2) = \sum_{\{i,j\} \in P} \overline{K}_{i,j}(\sigma_1, \sigma_2).$$

8.2.3 When a Distance Measure is a metric

A binary function d is called symmetric if $d(x, y) = d(y, x)$ for all x, y in the domain, and is called regular if $d(x, y) = 0$ if and only if $x = y$. We define a distance measure to be a nonnegative, symmetric, regular binary function. A metric is a distance measure d that satisfies the triangle inequality $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, z in the domain.

8.3 Distance Measures for Lists of XML Trees

In this section, we first provide some definitions (Section 8.3.1) then present the *XLS* measure (Section 8.3.2), *XLS-P* measure (Section 8.3.3 3.3) and finally the *XLS-PP* measure (Section 8.3.4). Normalization issues are discussed in Section 8.5.

8.3.1 Problem Definition

The goal of this work is to define and compute the distance between two lists La, Lb of XML trees, $La = Ta_1, Ta_2 \cdots Ta_k$ and $Lb = Tb_1, Tb_2 \cdots, Tb_k$, where Tx_i are XML trees. Often, as is the case with XML proximity search systems, all Ta_i, Tb_j are included (obtained by a sequence of deletes) in a tree Ti of a collection $D = T1, \cdots, Tn$. However, this property is not important in our definitions. Note that for the case of complete lists (permutations) of subtrees where each subtree appears in both lists, the problem is reduced to the permutations distance problem which we discussed in Section 8.2.2. However, this case is not practical since XML search engines return different XML trees. Hence, we focus on top-k lists.

A *total mapping* f from La to Lb is a bijection from La to Lb. Hence, tree Ta_i is mapped to $Tb_j = f(Ta_i)$. Let N be the set of all possible total mappings, f from La to Lb. Similarly, a partial mapping g is a partial function from La to Lb.

Let $TS(T1, T2)$ be the tree similarity between two trees $T1$, $T2$. TS can be the tree edit distance or another measure as discussed in Section 8.2. TS is normalized in [0,1] as explained in Section 8.5.1.

8.3.2 XML Lists Similarity based on Total Mapping

In this section we present our first measure for the distance between two top-k lists of XML trees. The key intuition is that we extend previous list distance measures that only consider exact mappings between the objects of the two lists to also consider approximate mappings. In particular, we first compute the closest pairwise mappings between the XML trees from the two lists and then view these mappings as exact mappings and apply list permutation distance measures.

Assuming k elements in each XML list, *XLS* is defined as follows. First we define the total mapping similarity distance $MSD^T(La, Lb, f)$ between La and Lb for a total mapping f as

$$MSD^T(La, Lb, f) = \frac{\sum_{i=1\cdots k} TS(Ta_i, f(Ta_i))}{k} \tag{8.1}$$

That is, MSD^T is a measure of how "tight" the total mapping f is. Notice that $MSD^T(La, Lb, f)$ takes values in [0,1], since TS is also in [0,1] and we divide by k.

We next define the minimum total mapping $fmin^T$ as the total mapping between La and Lb with minimum $MSD^T(La, Lb, f)$. It is,

$$fmin^T = argmin_f MSD^T(La, Lb, f) \tag{8.2}$$

that is, $argmin_f$ is the f that minimizes MSD^T.

Given $fmin^T$, we define the minimum total mapping similarity distance,

$$MinMSD^T(La, Lb) = MSD^T(La, Lb, fmin^T) \tag{8.3}$$

Definition 1: The XML Lists Similarity based on total mapping (XLS) between XML lists La, Lb is the minimum total mapping similarity distance. It is:

$$XLS(La, Lb) = MinMSD^T(La, Lb) \tag{8.4}$$

Notice that $XLS(La, Lb)$ is in [0,1] since $MinMSD^T(La, Lb)$ is in [0,1].

Measures for $MinMSD^T(La, Lb)$: The tree similarity, TS which is used to compute $MinMSD^T(La, Lb)$ can be any of the tree or XML similarity measures discussed in Section 8.2.1. The only constraint (as we show in Theorem 8.3.1) is that the measure used must be a metric if XLS is to be a metric.

Theorem 8.3.1 *XLS is a metric if the tree similarity measure employed, TS, is a metric.*

Proof: It is straightforward that XLS is nonnegative ($XLS(La, Lb) \geq 0$), symmetric ($XLS(La, Lb) = XLS(Lb, La)$) and regular ($XLS(La, La) = 0$) since this holds for tree similarity measure, TS which is a metric.

We need to prove the triangular property, that is, for any tree lists La, Lb, Lc prove that:

$$XLS(La, Lc) \leq XLS(La, Lb) + XLS(Lb, Lc) \tag{8.5}$$

To do so, we will prove the triangular property for $MinMSD^T(\cdot, \cdot)$. That is we need to prove that:

$$MinMSD^T(La, Lc) \leq MinMSD^T(La, Lb) + MinMSD^T(Lb, Lc) \tag{8.6}$$

Prove triangular property for MinMSDT: From Equations 8.1 and 8.3 (we skip k in denominator of Equation 8.1 throughout the proof, as it is for normalization purposes and does not affect the proof correctness):

$$\begin{aligned} MinMSD^T(La, Lb) &= MSD^T(La, Lb, fmin^T_{ab}) \\ &= \sum_{i=1\cdots k} TS(Ta_i, fmin^T_{ab}(Tb_i)) \end{aligned} \tag{8.7}$$

where $fmin_{ab}$ is the minimum total mapping from La to Lb.

Similarly:

$$\begin{aligned} MinMSD^T(Lb, Lc) &= MSD^T(Lb, Lc, fmin^T_{bc}) \\ &= \sum_{j=1\cdots k} TS(Tb_j, fmin^T_{bc}(Tb_j)) \end{aligned} \tag{8.8}$$

$$\begin{aligned} MinMSD^T(La, Lc) &= MSD^T(La, Lc, fmin^T_{ac}) \\ &= \sum i = 1 \cdots kTS(Ta_i, fmin^T_{ac}(Ta_i)) \end{aligned} \tag{8.9}$$

Hence, from Equations 8.7, 8.8 and 8.9, proving Equation 8.6 is equivalent to proving:

$$\begin{aligned} &\sum_{i=1\cdots k} TS(Ta_i, fmin^T_{ac}(Ta_i)) \leq \\ &\qquad \sum_{i=1\cdots k} TS(Ta_i, fmin^T_{ab}(Ta_i)) + \sum_{i=1\cdots k} TS(Tb_j, fmin^T_{bc}(Tb_j)) \end{aligned} \tag{8.10}$$

Since the tree similarity measure $TS(\cdot,\cdot)$ is a metric, it satisfies the triangular property. Consider a tree Ta_i in La that is mapped to $Tb_j = fmin_{ab}^T(Ta_i)$ in Lb, which is in turn mapped to tree $Tc_s = fmin_{bc}^T(Tb_j) = fmin_{bc}^T(fmin_{ab}^T(Ta_i))$ in Lc. The triangular property for Ta_i, Tb_j, Tc_s can be written as:

$$TS(Ta_i, fmin_{bc}^T(fmin_{ab}^T(Ta_i))) \leq \\ TS(Ta_i, fmin_{ab}^T(Ta_i)) + TS(Tb_j, fmin_{bc}^T(Tb_j)) \quad (8.11)$$

Summing Equation 8.11 over all Ta_i's in La, and keeping in mind that $fmin_{ab}$, $fmin_{bc}$ are bijections, we get

$$\sum_{i=1\cdots k} TS(Ta_i, fmin_{bc}^T(fmin_{ab}^T(Ta_i))) \leq \\ \sum_{i=1\cdots k} TS(Ta_i, fmin_{ab}^T(Ta_i)) + \sum_{j=1\cdots k} TS(Tb_j, fmin_{bc}^T(Tb_j)) \quad (8.12)$$

The left hand side of Equation 8.12 is the total mapping similarity distance $MSD^T(La, Lc, f')$, where $f'(\cdot) = fmin_{bc}^T($
$fmin_{ab}^T(\cdot))$. We know from Equation 8.9, that $fmin_{ac}^T$ gives the minimum total mapping similarity distance between La, Lc. That is

$$MSD^T(La, Lc, fmin_{ac}^T) \leq MSD^T(La, Lc, f') \quad (8.13)$$

Hence,

$$\sum_{i=1\cdots k} TS(Ta_i, fmin_{ac}^T(Ta_i)) \leq \sum_{i=1\cdots k} TS(Ta_i, fmin_{bc}^T(fmin_{ab}^T(Ta_i))) \quad (8.14)$$

From Equations 8.11 and 8.14 we get Equation 8.10 which was our goal.

Note that Theorem 8.3.1 also applies for any XML similarity measure that is a metric, as explained in Section 8.4.

Example 1: *Consider the top-*3 *lists* La *and* Lb *in Figure 8.1. We will illustrate the steps involved in computing* $XLS(La, Lb)$*. In this example, we use tree edit distance,* TED *as the tree similarity measure,* TS*. We first compute the XML similarity component by finding all possible total mappings,* $N = \{f_1, f_2, f_3, f_4, f_5, f_6\}$*:*

$$f_1(Ta_1) = Tb_1, f_1(Ta_2) = Tb_2, f_1(Ta_3) = Tb_3$$
$$f_2(Ta_1) = Tb_3, f_2(Ta_2) = Tb_2, f_2(Ta_3) = Tb_1$$
$$f_3(Ta_1) = Tb_2, f_3(Ta_2) = Tb_1, f_3(Ta_3) = Tb_3$$
$$f_4(Ta_1) = Tb_1, f_4(Ta_2) = Tb_3, f_4(Ta_3) = Tb_2$$
$$f_5(Ta_1) = Tb_3, f_5(Ta_2) = Tb_1, f_5(Ta_3) = Tb_2$$
$$f_6(Ta_1) = Tb_2, f_6(Ta_2) = Tb_3, f_6(Ta_3) = Tb_1$$

The normalized tree edit distance (see Section 8.5.1) between each pair of trees in La and Lb is given by the following matrix:

$$\begin{array}{c|ccc} & Tb_1 & Tb_2 & Tb_3 \\ \hline Ta_1 & 0.00 & 0.78 & 0.71 \\ Ta_2 & 0.71 & 0.58 & 0.20 \\ Ta_3 & 0.78 & 0.43 & 0.58 \end{array}$$

The total mapping similarity distance of each total mapping in N is calculated by Equation 8.1 as follows:

$$MSD^T(La, Lb, f_1) = (0.00 + 0.58 + 0.58)/3 = 1.16/3 = 0.38$$
$$MSD^T(La, Lb, f_2) = (0.71 + 0.58 + 0.78)/3 = 2.07/3 = 0.69$$
$$MSD^T(La, Lb, f_3) = (0.78 + 0.71 + 0.58)/3 = 2.07/3 = 0.69$$
$$MSD^T(La, Lb, f_4) = (0.00 + 0.20 + 0.43)/3 = 0.63/3 = 0.21$$
$$MSD^T(La, Lb, f_5) = (0.71 + 0.71 + 0.43)/3 = 0.63/3 = 0.62$$
$$MSD^T(La, Lb, f_6) = (0.78 + 0.20 + 0.78)/3 = 0.63/3 = 0.59$$

Hence, f_4 is the mapping with the minimum mapping distance. It is $XLS(La, Lb) = minMSD^T(La, Lb) = MSD^T(La, Lb, f_4) = 0.21$.

8.3.3 XML Lists Similarity based on Total Mapping with Position Component

As we described in Section 8.3.2, XLS takes in to consideration the similarity of XML trees across each list. This works well in computing a reasonable similarity distance between top-k XML Lists where k is relatively small. When k is large, it is important to also take in the consideration, the position of the mapped trees in each list. For example, consider 3 top-k lists of XML trees La, Lb and Lc where $XLS(La, Lb) = XLS(Lb, Lc)$ but $fmin^T_{ab}$ preserves the correct order (i.e. $fmin^T_{ab}(Ta_1) = Tb_1$, $fmin^T_{ab}(Ta_2) = Tb_2$ and so on) while $fmin^T_{bc}$ maps the trees in reverse order (i.e. $fmin^T_{bc}(Tb_1) = Tc_k$, $fmin^T_{bc}(Tb_2) = Tc_{k-1}$ and so on). Ideally, we want the distance between La and Lb to be smaller than the distance between Lb and Lc. Hence, we define a measure, XML Lists Similarity based on Total Mapping with Position Component (XLS-P) that includes the mapping position distance in addition to the mapping similarity distance.

Definition 2: The XML Lists Similarity based on Total Mapping with Position Component (XLS-P) between XML lists La, Lb has two components:

- The XML similarity component $MinMSD^T(La, Lb)$.
- The total mapping position distance component $PD^T(La, Lb, fmin^T)$, which is also referred as the position component in this section. PD^T is defined using one of the well known metrics on permutations as discussed below. PD^T is in [0, 1] as discussed in Section 8.5.2. It is

$$XLS\text{-}P(La, Lb) = a \cdot MinMSD^T(La, Lb) + b \cdot PD^T(La, Lb, fmin^T) \qquad (8.15)$$

where a, b are the XML similarity and position component constants respectively. a, b adjust the relative importance of the two components. Notice that XLS-$P(La, Lb)$ is in [0,2] since $MinMSD^T(La, Lb)$ and $PD^T(La, Lb, fmin^T)$ are in [0,1] and constants a and b are in [0,1].

We choose $fmin^T$ to minimize the XML similarity component and not the whole XLS-P, because we believe it is more intuitive to compute the distance component based on the tightest XML similarity mapping rather than mixing the two components.

Note that other functions can be used to combine the contribution of the two components, as we discuss below.

Measures for XML Similarity component,

$MinMSD^T(La, Lb)$**:** The tree similarity, TS which is used to compute $MinMSD^T(La, Lb)$ can be any of the tree or XML similarity measures discussed in Section 8.2.1.

Measures for Position component, $PD^T(La, Lb, fmin^T)$**:** Note that list permutation distance metrics (not top-k list distance measures) are used in $XLS\text{-}P$. Given the mapping $fmin^T$, we naturally extend the Spearman's footrule distance and Kendall tau distance for permutations with ties [Dia88, FKM+04, FKS03, KG90] as follows:

Position distance (PD^{TF}) based on Spearman's footrule metric for permutations, is given by:

$$PD^{TF}(La, Lb, fmin^T) = \sum_{i=1}^{k} \left| pos_{La}(Ta_i) - pos_{Lb}(fmin^T(Ta_i)) \right| \tag{8.16}$$

where $pos_{La}(Ta_i)$ is the position of tree Ta_i in list La. This formula is extended as follows to consider ties. A set of trees with the same score is called a bucket. The ranked list of results can be then viewed as ranked list of buckets $B_1, B_2, \cdots, B_n$. The position of bucket B_i, denoted $pos(B_i)$ is the average result location within bucket B_i. We assign $pos_{La}(Ta_i) = pos(B(Ta_i))$ where $B(Ta_i)$ is the bucket of Ta_i.

Position distance ($PDTK$) based on Kendall tau metric for permutations considering ties, is given by:

$$PD^{TK}(La, Lb, fmin^T) = \sum_{\{i,j\} \in P} \overline{K}_{i,j}(La, Lb') \tag{8.17}$$

where Lb' is constructed from list Lb when element Tb_j is replaced by $Ta_i = (fmin^T)^{-1}(Tb_j)$, that is, $Tb_j = fmin^T(Ta_i)$. That is, we assume that an element Ta_i in La and its corresponding element Tb_j in Lb are the same. Hence, we just have k distinct elements $1, 2, \cdots, k$ in both lists, and the problem of computing $PD^{TK}(La, Lb, fmin^T)$ of the two XML lists is same as computing the Kendall Tau metric of two permutations. P is the set of all unordered pairs of the k distinct elements.

Hence, there are two variants of $XLS\text{-}P$:

$$XLS\text{-}P^F(La, Lb) = a \cdot MinMSD^T(La, Lb) + b \cdot PD^{TF}(La, Lb, fmin^T) \tag{8.18}$$

$$XLS\text{-}P^K(La, Lb) = a \cdot MinMSD^T(La, Lb) + b \cdot PD^{TK}(La, Lb, fmin^T) \tag{8.19}$$

$XLS\text{-}P$ **is not a metric:** The XML Lists Similarity based on Total Mapping with Position Component ($XLS\text{-}P$) is not a metric because the total mapping position distance component $PD^T(La, Lb, fmin^T)$ is not a metric. In particular, $PD^T(La, Lb, fmin^T)$ does not satisfy the triangular inequality property. This is because the mapping $fmin^T$ is computed by comparing XML trees (accounting for possible tree overlaps) and not by comparing whole objects. To be more specific, if we consider three lists of (whole) objects Wa, Wb and Wc, then $f^T_{ac}(\cdot) = f^T_{bc}(f^T_{ab}(\cdot))$ (where f^T_{ac} is a total mapping between Wa and Wc) since we can only have "exact" matches. But if we consider three lists of XML trees La, Lb and Lc, typically $fmin^T_{ac}(\cdot) \neq fmin^T_{bc}(fmin^T_{ab}(\cdot))$ since we could have "partial" matches. The following example illustrates this scenario:

Let La, Lb and Lc be the following top-2 lists of XML trees. $La = (Ta_1, Ta_2)$, $Lb = (Tb_1, Tb_2)$ and $Lc = (Tc_1, Tc_2)$. Now, suppose that $TS(Ta_1, Tb_1) = TS(Ta_2, Tb_2) = TS(Tb_1, Tc_1) = TS(Tb_2, Tc_2) = TS(Ta_1, Tc_2) = TS(Ta_2, Tc_1) = 0.4$ and all other distances (between the remaining pairs across the different lists) are 0.6 (and for all x, $TS(x, x) = 0$). Then, the following would be the minimum total mappings between each list La, Lb and Lc:

$$fmin^T_{ab}(Ta_1) = Tb_1,\ fmin^T_{ab}(Ta_2) = Tb_2\ fmin^T_{bc}(Tb_1) = Tc_1,\ fmin^T_{bc}(Tb_2) = Tc_2$$
$$fmin^T_{ac}(Ta_1) = Tc_2,\ fmin^T_{ac}(Ta_2) = Tc_1$$

If we assume $a = 1$ and $b = 1$, then $minMSD^T(La, Lb) = minMSD^T(Lb, Lc) = minMSD^T(La, Lc) = 0.4 + 0.4 = 0.8$. But, $fmin^T_{ab}$ and $fmin^T_{bc}$ preserve order (i.e., Ta_1 is mapped to Tb_1, Ta_2 is mapped to Tb_2 and so on.), but $fmin^T_{ac}$ does not preserve order (it maps Ta_1 to Tc_2 and Ta_2 to Tc_1). Hence we have
$PD^T(La, Lb, fmin^T_{ab}) = PD^T(La, Lb, fmin^T_{bc}) = 0.0$ and $XLS\text{-}P(La, Lb) = XLS\text{-}P(Lb, Lc)$. Now, since $fmin^T_{ac}$ does not preserve order, $PD^T(La, Lb, fmin^T_{ab}) > 0$ (in fact the actual value would be 1.0 as it maps the elements in reverse order). So, $XLS\text{-}P(La, Lc) = 0.4 + 1.0 = 1.4$. This breaks the triangular inequality property since $XLS\text{-}P(La, Lb) + XLS\text{-}P(Lb, Lc) = 0.4 + 0.4 = 0.8 > 1.4$.

Example 1 (cont'd): *Consider the top-3 lists La and Lb in Figure 8.1. We will illustrate the steps involved in computing $XLS\text{-}P(La, Lb)$. As before, we first compute f_4 –the total mapping with the minimum mapping similarity distance. It is $minMSD^T(La, Lb) = MSD^T(La, Lb, f_4) = 0.21$. The normalized Spearman's footrule position component is $PD^{TF}(La, Lb, f_4) = 2.0/4.0 = 0.5$. Hence, $XLS\text{-}PF(La, Lb) = 0.21 + 0.5 = 0.71$ (assuming $a = 1$ and $b = 1$). If the position distance is calculated using normalized Kendall tau, then $PD^{TK}(La, Lb, f_4) = 1.0/3.0 = 0.33$ and $XLS\text{-}PK(La, Lb) = 0.21 + 0.33 = 0.54$ (assuming $a = 1$ and $b = 1$). The difference in the two scores is due to inherent differences between the Spearman's footrule and Kendall tau metrics.*

8.3.4 XML Lists Similarity based on Partial Mapping with Position Component

The total mapping distance measures in Section 8.3.2 have the drawback that two totally irrelevant trees from the two lists may be mapped to each other, given that all trees must be mapped between the two lists. This is unintuitive and may lead to confusing results, especially for the positional component of the measure. To overcome this drawback, we propose the *partial mapping* measures, where trees from the two lists are mapped only if they are adequately similar.

Similarity Threshold: In order to partially map the two lists of XML trees, we specify a threshold ω, which is set to a value in [0, 1]. Intuitively, we only create mappings between trees of the two lists whose tree similarity (TS) is up to ω. For example, if we want to create only the mappings between trees that are at most 40% different, then we set $\omega = 0.4$. Notice that TS is also in [0,1] as described in Section 8.5.1. The threshold ω is chosen given the application's characteristics. We consider various values for ω in Section 8.8. Note that for $\omega = 1$, $XLS\text{-}PP$ reduces to $XLS\text{-}P$.

Assuming k elements in each XML list, $XLS\text{-}PP$ is defined as follows. First we define the partial mapping g for a total mapping f and threshold ω.g is a partial function defined only for XML trees Ta_i with $TS(Ta_i, f(Ta_i)) \leq \omega$. Then $g(Ta_i) = f(Ta_i)$. Let La^g be the subset of La that contains the XML trees that have a mapping for g.

Next we define the *partial mapping similarity distance* $MSD^P(La, Lb, g)$ between La and Lb given a partial mapping g as:

$$MSD^P(La, Lb, g) = \frac{\sum_{Ta_i \in La^g} TS(Ta_i, g(Ta_i)) + \sum_{Ta_i \in \{La - La^g\}} c}{k \cdot max(c, \omega)} \tag{8.20}$$

where XML trees that do not get mapped incur a penalty cost, c. Notice that $MSD^P(La, Lb, g)$ is also in [0,1] since TS is in [0,1] and we divide by $k \cdot max(c, \omega)$. Note that penalty cost, c is also in [0,1].

We next define the minimum partial mapping $gmin^P$ between La and Lb given a threshold, ω as the partial mapping that has a corresponding total mapping f for threshold ω and has the minimum $MSD^P(La, Lb, g)$. That is,

$$gmin^P = argmin_g MSD^P(La, Lb, g) \tag{8.21}$$

We emphasize that g must come from a total mapping, in order for the metric properties defined below to hold.

Given $gmin^P$, we define the *minimum partial mapping similarity distance*

$$MinMSD^P(La, Lb) = MSD^P(La, Lb, gmin^P) \tag{8.22}$$

Definition 3: The *XML Lists Similarity based on Partial Mapping with Position Component (XLS-PP)* has two components:

a The *XML partial similarity component* $MinMSD^P(La, Lb)$.

b The *partial mapping position distance component* $PD^P(La, Lb, gmin^P)$, which is also referred as the position component. PD^P can be one of the well known measures (some are not metrics) on top-k lists as discussed below. PD^P is in [0,1] as discussed in Section 8.5.2.

It is:

$$XLS\text{-}PP(La, Lb) = a \cdot MinMSD^P(La, Lb) + bPD^P(La, Lb, gmin^P) \tag{8.23}$$

where a, b are constants defined as in Section 8.3.2. Notice that $XLS\text{-}PP(La, Lb)$ is in [0,2] since $MinMSD^P(La, Lb)$ and $PD^P(La, Lb, gmin^P)$ are in [0,1] and constants a and b are in [0,1].

The same tree similarity measures as in XLS can be used for the XML partial similarity component.

Measures for Position component, $PD^P(La, Lb, gmin^P)$**:** We need to use partial (top-k) list distance measures. Given the partial mapping $gmin^P$, we naturally extend the Spearman's footrule distance and Kendall tau distance for top-k lists with ties by combining previous works [Dia88, FKM$^+$04, FKS03, KG90], which separately tackle the top-k [FKS03] and the ties [FKM$^+$04] issues, as follows:

Position distance $PD^{PF(l)}$ based on Spearman's footrule for top-k lists with location parameter l considering ties is computed as follows. We place all trees in both lists whose tree similarity TS is greater than threshold ω at position l. Let list Lb be a list constructed by Lb by replacing each element Tb_i by $Ta_j = (gmin^P) - 1(Tb_i)$, if this mapping exists (recall that $gmin^P$ is a partial function). Then,

$$PD^{PF(l)}(La, Lb, gmin^P) = F^{(l)}(La, Lb') \tag{8.24}$$

where $F^{(l)}(\cdot, \cdot)$ is the footrule function for top-k lists defined in [FKS03]. We extend this formula to consider ties by considering buckets for computing the position as explained in Section 8.3.2.

Position distance $PD^{PK(p)}(La, Lb, gmin^P)$ based on Kendall tau metric for top-k lists with penalty parameter p, considering ties, is given by:

$$PD^{PK(p)}(La, Lb) = \sum_{\{i,j\} \in La \cup Lb'} \overline{K}^{(p)}_{(i,j)}(La, Lb') \tag{8.25}$$

where Lb' is defined as in Section 8.3.2, and is defined as in [FKS03].

Hence, we have two variants of $XLS\text{-}PP$:

$$XLS\text{-}PP^F(La, Lb) = a \cdot MinMSD^P(La, Lb) + b \cdot PD^{PF(l)}(La, Lb, gmin^P) \tag{8.26}$$

$$XLS\text{-}PP^K(La, Lb) = a \cdot MinMSD^P(La, Lb) + b \cdot PD^{PK(p)}(La, Lb, gmin^P) \tag{8.27}$$

XLS-PP is not a metric: XLS-PP is not a metric because MSD^P is not a metric. The reason is that the triangular property does not hold for any choices of threshold, ω (in [0,1]) and penalty constant, c (in [0,1]).

Example 1 (cont'd): *Consider again the lists La and Lb in Figure 8.1. Assuming $\omega = 0.4$ and $c = 0.4$, we get the following partial mappings from the previous total mappings:* $g_1(Ta_1) = Tb_1$.
g_2, g_3 are empty mappings.
$g_4(Ta_1) = Tb_1$, $g_4(Ta_2) = Tb_3$. g_5 is again an empty mapping.
$g_6(Ta_2) = Tb_3$.

The mapping distance of each partial mapping is as follows:

$$\begin{aligned}
MSD^P(La, Lb, g_1) &= (0.00 + c + c)/(3 \cdot max(c, \omega)) \\
&= (0.00 + 0.40 + 0.40)/(3 \cdot 0.40) \\
&= 0.80/1.2 = 0.66 \\
MSD^P(La, Lb, g_2) &= (c + c + c)/(3 \cdot max(c, \omega)) \\
&= (0.40 + 0.40 + 0.40)/(3 \cdot 0.40) \\
&= 1.20/1.20 = 1.00 \\
MSD^P(La, Lb, g_3) &= 1.00 \\
MSD^P(La, Lb, g_4) &= (0.00 + 0.20 + c)/(3 \cdot max(c, \omega)) \\
&= (0.00 + 0.20 + 0.40)/(30.40) \\
&= 0.60/1.2 = 0.50 \\
MSD^P(La, Lb, g_5) &= 1.00 \\
MSD^P(La, Lb, g_6) &= (c + 0.20 + c)/(3 \cdot max(c, \omega)) \\
&= (0.40 + 0.20 + 0.40)/(30.40) \\
&= 1.00/1.2 = 0.83
\end{aligned}$$

g_4 is the mapping with the minimum mapping distance. $minMSD^P(La, Lb) = MSD^P(La, Lb, g_4) = 0.50$.

The Spearman's footrule position component $PD^{PF}(La, Lb, g_4) = 4.0/12.0 = 0.33$. $XLS\text{-}PP^F(La, Lb) = 0.50 + 0.33 = 0.83$ (assuming $a = 1$ and $b = 1$).

If the position distance is calculated using normalized Kendall tau, then $PD^P(La, Lb, g_4) = 2.0/12.0 = 0.17$. $XLS\text{-}PP^K(La, Lb) = 0.50 + 0.17 = 0.67$ (assuming $a = 1$ and $b = 1$).

Notice that the normalized position component in $XLS\text{-}P$ is smaller than in $XLS\text{-}PP$, even though two trees do not match in $XLS\text{-}PP$. The reason is that the maximum value (used in normalizing as we describe in Section 8.5.2) of position distance (PDP) is larger in $XLS\text{-}PP$.

8.4 XML Similarity Measures for Various Tree Similarity Measures

As mentioned before, only those tree similarity measures that are metrics may lead to a distance metric for XML lists. In particular, if the tree similarity measure is a metric, then

- XLS is a metric (as proved in Section 8.3.2);
- $XLS\text{-}P$ and $XLS\text{-}PP$ are not metrics (as proved in Sections 8.3.3 and 8.3.4).

The following tree similarity measures are metrics: Tree-Edit Distance [Bil05], Tree-Edit-based Structural Distance [NJ02], Fourier Transform-based Distance [FMPP02, FMMP05], Entropy-based Similarity [Hel07], and the similarity measure in [LCS$^+$04]. In Table 8.1 we present these results in more detail along with the complexity of calculating each tree similarity measure.

Theorem 8.4.1 *Tree edit distance is a distance metric*

Proof: Following [Bil05], we assume throughout the paper that labels assigned to nodes are chosen from a finite alphabet Σ. Let $\lambda \notin \Sigma$ denote a special blank symbol and define $\Sigma_\lambda = \Sigma \cup \lambda$. We define a cost function $\gamma : (\Sigma_\lambda \times \Sigma_\lambda) \| (\lambda, \lambda) \to R$, on pairs of labels. We will always assume that γ is a distance metric. That is, for any $l_1, l_2, l_3 \in \Sigma$ the following conditions are satisfied:

1. $\gamma(l_1, l_2) \geq 0$ (non-negative), $\gamma(l_1, l_1) = 0$ (regular).
2. $\gamma(l_1, l_2) = \gamma(l_2, l_1)$ (symmetric).

Table 8.1: Tree Similarity Measures and Their Properties.

Tree Similarity Measure	**Is a Metric?**	**XML-Specific?**	**Time Complexity to compute the measure between pair of trees**
Tree-Edit Distance [Bil05]	Yes	No	$O(Cost(TD(Ta_i, Tb_j)) = O(\|Ta_i\| \cdot \|Tb_j\| \cdot min(leaves(Ta_i), depth(Ta_i)) \cdot min(leaves(Tb_j), depth(Tb_j))$ [NJ02]
Tree-Alignment Distance [Bil05, JWZ94]	No (fails triangle-inequality)	No	$O(\|Ta_i\| \cdot \|Tb_j\| \cdot (deg(Ta_i) + deg(Tb_j))^2)$
Tree-Edit based Structural Distance [NJ02]	Yes	Yes	$O(\|Ta_i\| \cdot \|Tb_j\|)$
Fourier Transform-based Distance [FMPP02, FMMP05]	Yes	Yes	$O(NlogN)$ where $N = max(\|Ta_i\| \cdot \|Tb_j\|)$
Entropy-based Similarity [Hel07]	Yes	Yes	$O(N)$ where $N = max(\|Ta_i\| \cdot \|Tb_j\|)$
Path-Shingle based Similarity [But04]	No (since it is based on hashing)	Yes	Linear time in general. $O(\|Ta_i\| + \|Tb_j\|)$
Similarity measure in [LCS$^+$04]	Yes	Yes	Linear time in general. $O(\|Ta_i\| + \|Tb_j\|)$
Similarity measure in [YKT05]	No	No	Linear time in general. $O(\|Ta_i\| + \|Tb_j\|)$

3. $\gamma(l_1, l_3) \leq \gamma(l_1, l_2) + \gamma(l_2, l_3)$ (triangle inequality).

Let T_1 and T_2 be labeled trees. We represent each edit operation by $(l_1 \to l_2)$, where $(l_1, l_2) \in (\Sigma_\lambda \times \Sigma_\lambda) \setminus (\lambda, \lambda)$. The operation is a relabeling if $l_1 \neq \lambda$ and $l_2 \neq \lambda$, a deletion if $l_2 = \lambda$, and an insertion if $l_1 = \lambda$. We extend the notation such that $(v \to w)$ for nodes v and w denotes $(label(v) \to label(w))$. Here, as with the labels, v or w may be λ. Given a metric cost function ? defined on pairs of labels we define the cost of an edit operation by setting $\gamma(l_1 \to l_2) = \gamma(l_1, l_2)$. The cost of a sequence $S = s1, \cdots, sk$ of operations is given by $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$.

The edit distance, $TED(T_1, T_2)$, between T_1 and T_2 is formally defined as:

$TED(T_1, T_2) = min\{\gamma(S) \mid S$ *is a sequence of operations transforming* T_1 *into* $T_2\}$.

Since γ is a distance metric, TED also becomes a distance metric as follows:

$TED(T_1, T_2) > 0$ (*non-negative*). Each tree edit operation has a non-negative cost and hence their summation would also be non-negative, because γ is a distance metric.

$TED(T_1, T_1) = 0$ (*regular*) as no tree-edit operations are required to transform a tree to itself and hence the cost is 0.

$TED(T_1, T_2) = TED(T_2, T_1)$ (*symmetric*). Let $s_1, \cdots, s_k$ be the sequence of edit operations to transform Ta to Tb. Then, we can transform Tb to Ta by sequence $s'_k, \cdots, s'_1$, where s'_i is the dual of s_i. For e.g., if the edit operation s_i adds node v, s'_i would remove node v and so on. This would generate a sequence of edit operations that transforms T_2 to T_1 with minimum cost, because if there were a sequence of operations that transforms T_2 to T_1 with cost lesser than this, then, using that sequence, we could obtain a sequence of operations that transforms T_1 to T_2 with the same cost, which is a contradiction. Note that for the symmetricity property to hold, an operation and its dual –both should be assigned the same penalty.

$TED(T_a, T_c) < TED(T_a, T_b) + TED(T_b, T_c)$ (*triangle inequality*). Let S_1 be the sequence of operations that transforms Ta to Tb and S_2 be the sequence that transforms Tb to Tc. Then the sequence $S_3 = concatenate(S_1, S_2)$ can transform Ta to Tc and $\gamma(S_3) = \gamma(S_1) + \gamma(S_2)$, which proves the triangle inequality. If there is another sequence S'_3 that goes from Ta to Tc with $\gamma(S'_3) < \gamma(S_3)$ (since the tree edit distance is the minimum distance as mentioned above) then it will be $\gamma(S'_3) \leq \gamma(S_1) + \gamma(S_2)$.

8.5 Normalization

In this section we discuss how we normalize the XML similarity component and the position components of XLS-P and XLS-PP, in Sections 8.3.3 and 8.3.4 respectively. Normalization of XML similarity component depends on the tree similarity measure (TS) employed. In Section 8.5.1, we discuss the normalization steps when tree edit distance (TED) is used as the tree similarity measure. Section 8.5.2 discusses the normalization of the position component.

8.5.1 Normalize Tree-Edit Distance based XML Similarity Component

Let T_1 and T_2 be two rooted, ordered and labeled XML trees And let $TED(T_1, T_2)$ be the tree edit distance between T_1, T_2. Let $TED_{max}(T_1, T_2)$ be the maximum cost among the costs of all possible sequences of tree-edit operations that transform T_1 to T_2 (notice that the tree edit distance, $TED(T_1, T_2)$ is the minimum cost among the costs of all possible sequences of tree-edit operations). We normalize the tree edit distance by dividing the tree edit distance, $TED(T_1, T_2)$ by $TED_{max}(T_1, T_2)$.This normalized $TED(T_1, T_2)$ is also called Structural Distance in [DCWS04, DCjWS06]. To calculate $TED_{max}(T_1, T_2)$, we calculate the cost to delete all nodes from T_1 and insert all nodes from T_2. That is, $TED_{max}(T_1, T_2) = size(T_1) \cdot D_p + size(T_2) \cdot I_p$ where D_p and I_p are the delete and insert penalties and $size(T_1)$ is the number of nodes present in tree T_1.

We use unit delete and insert penalties in our experiments. The normalized $TED(T_1, T_2)$ is low when the trees have similar structure and high percentage of matching nodes, and high when the trees have different structure and low percentage of matching nodes (0 [1] is the min [max] value).

8.5.2 Normalize Position Component

In $XLS\text{-}P$**:** To normalize the position component in $XLS\text{-}P$, we refer to the metrics on permutations presented in [FKS03]. The maximum value of $PD^{TK}(La, Lb, f)$ is $k(k-1)/2$, which occurs when La is the reverse of Lb. The maximum value of $PD^{TF}(La, Lb, f)$ is $k2/2$ when k is even and $(k+1)(k-1)/2$ when k is odd. As with Spearman's footrule, the maximum occurs when La is the reverse of Lb. Hence, to normalize we divide the metrics by these maximum values.

In $XLS\text{-}PP$**:** To normalize the position component in $XLS\text{-}PP$, we refer to the metrics on top-k lists presented in [FKS03]. In order to normalize the position components of two top-k lists, we divide them by their maximum values which occur when there are no mappings between Lists La and Lb.

Theorem 8.5.1 *The maximum value of top-k Spearman's footrule $PD^{PF(l)}(La, Lb)$ is $2k(l-(k+1)/2)$ where l is the location parameter.*

Proof: Since there are no mappings between the top-k lists, all k elements of each of the list get mapped to location l. Hence,

$$\begin{aligned} PR^{F(l)}_{max}(La, Lb) &= 2(|1-l| + |2-l| + \cdots + |k-l|) \\ &= 2k(l-(k+1)/2) \end{aligned}$$

For a natural choice of $l = k+1$, the maximum value is $k(k+1)$, which we use in our experiments.

Theorem 8.5.2 *The maximum value of top-k Kendall tau, $PR^{k(p)}(La, Lb)$ is $pk(k-1)+k^2$ where p is the penalty parameter.*

Proof: Since there are no mappings between the top-k lists, there are $2k$ distinct elements in $La \cup Lb$. For the unordered pairs within each list, $\bar{K}^{(}_{(i,j)}p)(La, Lb') = p$ since these pairs do not appear in the other list. There are $k(k-1)/2$ such pairs and considering both the lists, there are $k(k-1)$ such pairs, each with penalty p. Hence the total penalty is $pk(k-1)$. For the unordered pairs across each of the two lists, $\bar{K}^{(}_{(i,j)}p)(La, Lb') = 1$ since one element in each pair does not appear in the other list. There is k^2 such pairs, each with penalty 1. Hence the total penalty in this case is k^2. Adding them together, we get the maximum value which is $pk(k-1)+k^2$.

We use $p = 0.5$ in our experiments.

8.6 Algorithms

In this section, we describe efficient algorithms to compute XLS (Section 8.6.1), XLS-P (Section 8.6.2) and XLS-PP (Section 8.6.3) given two XML top-k lists.

8.6.1 Compute XLS

In this section, we describe efficient algorithms to compute XLS given two XML top-k lists.

Naïve approach: XLS-P for any two top-k XML lists La and Lb is computed as follows. First, the set N of all possible total mappings from La to Lb is computed. Then, for each total mapping f in N, we compute the total mapping similarity distance, $MSD^T(La, Lb, f)$ using Equation 8.1, and then find the minimum mapping $fmin^T$. Then, we compute XLS-$P(La, Lb)$ using Equation 8.4.

Overview of our algorithm: Instead of computing the set N of all possible total mappings and then selecting the minimum mapping $fmin^T$, we pre-compute the tree similarity measure of each tree pair across the two lists, build a bipartite graph, and apply a minimum cost perfect matching algorithm (we use the Hungarian algorithm [Mun57]) to compute all minimum mappings $fmin^T$. This procedure is presented in Algorithm 8.1.

Algorithm details: The following high level steps of execution explain the algorithm in detail:

1. Pre-compute the tree similarity $TS(Ta_i, Tb_j)$ between every pair of XML trees, one from each list La and Lb. There are k^2 such pairs, hence the complexity of this step is $k^2 \cdot Cost(TS(Ta_i, Tb_j))$ where $Cost(TS(Ta_i, Tb_j))$ is the complexity of computing the tree similarity between the two trees Ta_i and Tb_j. We use the dynamic programming algorithm by Zhang and Shasha [ZS89] to compute the edit-distance between ordered trees [Bil05]

Listing 8.1: Algorithm for computing XLS.

```
procedure ComputeXLS($La = \{Ta_1, Ta_2, \cdots, Ta_k\}$,
   $Lb = \{Tb_1, Tb_2, \cdots, Tb_k\}$, int $a$, int $b$)
begin
  $S[k, k] \leftarrow$ 2-D array that stores the tree similarity
     measures between every pair of XML trees (one from
     each List)
  for $i$ in $1, \cdots, k$
    for $j$ in $1, \cdots, k$
      Compute $TS(Ta, Tb)$; // Section 8.2.1
      Normalize $TS(Ta_i, Tb_j)$; // Section 8.5.1
      $S[i, j] \leftarrow TS(Ta_i, Tb_j)$
    end
  end
end
```

(any available algorithm can be employed to compute tree edit distance) as it is a popular tree-edit distance algorithm also available online . We refer to a detailed survey of tree edit distance algorithms [Bil05].

2. Create a weighted complete bipartite graph $G(C, P, W)$ as follows. The first set of nodes $C = 1, 2, \cdots, k$ denote the set of elements in XML list La. The second set of nodes $P = 1, 2, \cdots, k$ denote the set of elements in XML list Lb. The weight $W(i, j) = TS(Ta_i, Tb_j)$.In this section, we describe efficient algorithms to compute XLS-P given two XML top-k lists.

3. Execute a minimum cost perfect matching algorithm on $G(C, P, W)$ to compute $fmin^T$. We use the Hungarian algorithm [Mun57]. Finally, XLS is computed using Equation 8.4. The complexity of the Hungarian algorithm is $O(k^3)$.

4. Total Complexity of the algorithm is
   $O(k^2 \cdot Cost(TS(Ta_i, Tb_j)) + k^3)$.

Listing 8.2: Algorithm for computing $XLS\text{-}P$.

```
procedure ComputeXLS-P($La = \{Ta_1, Ta_2, \cdots, Ta_k\}$,
    $Lb = \{Tb_1, Tb_2, \cdots, Tb_k\}$, int $a$, int $b$)
begin
  /* Replace Line 9 in Listing 8.1 with: */
  Compute $PDT(La, Lb, fmin^T)$ using Eq. 8.16 (for Spearman's
      footrule) or Eq. 8.17 (for Kendall Tau)
  Compute $XLS\text{-}P$ using Eq. 8.18 or Eq. 8.19
end
```

8.6.2 Compute *XLS-P*

This algorithm is similar to the algorithm in Section 8.6.1, except for a few changes as we will describe. $fmin^T$ is computed as before and then the position distance $PD^T(La, Lb, fmin^T)$ is computed using Equations 8.16 and 8.17) for Spearman's footrule and Kendall tau position component respectively. Then, $XLS\text{-}P$ is computed using Equation 8.18 or 8.19.

Total Complexity of the algorithm is $O(k^2 \cdot Cost(TS(Ta_i, Tb_j)) + k^3 + k^2)$. Note the additional $O(k^2)$ to compute the position component.

8.6.3 Compute *XLS-PP*

In this section, we describe efficient algorithms to compute $XLS\text{-}PP$ given two XML top-k lists.

Naïve approach: $XLS\text{-}PP$ for two top-k XML lists La and Lb is computed as follows –given a threshold, ω and penalty constant, c: First, the set N of all possible total mappings from La to Lb is computed. Then, for each total mapping f in N, we compute a partial mapping g by retaining only those mapping instances in f whose tree similarity, $TS(\cdot, \cdot)$ between the corresponding pair of trees is at least ω. Then, for each g we compute the partial mapping similarity distance, $MSD^P(La, Lb, g)$ using Equation 8.20 and then find the minimum mapping $gmin^P$. Then we compute the position distance, $PD^P(La, Lb, gmin^P)$ (using Equation 8.24 or 8.25). Finally, we compute $XLS\text{-}PP(La, Lb)$ using Equation 8.26 or 8.27.

Listing 8.3: Algorithm for computing XLS-PP.

```
procedure ComputeXLS-PP($La = \{Ta_1, Ta_2, \cdots, Ta_k\}$,
    $Lb = \{Tb_1, Tb_2, \cdots, Tb_k\}$, int $\omega$, int $c$, int $a$, int $b$)
begin
  /* Replace Line 6 in Listing 8.2 with the following:
  if $TS(Ta_i, Tb_j) \leq \omega$
     $S[i, j] \leftarrow TS(Ta_i, Tb_j)$
  else
     $S[i, j] \leftarrow \infty$
  end if
  /* Replace Line 7 with the following: */
  $assignment_m[k, 2] \leftarrow$ 2-D array that stores the $m^{th} gmin^P$ with
      the minimum mapping distance
  /* Replace Line 9 with the following: */
  Compute $PDP(La, Lb, gmin^P)$ using Equation 8.24 (for
     Spearman's footrule) or Equation 8.25 (for Kendall
     Tau)
  /* Replace Line 10 with the following: */
  Compute $XLS$-$PP$ using Equation 8.26 or 8.27
end
```

Our algorithm: This algorithm is similar to the algorithm in Section 8.6.1, except for a few changes as we will describe. In Step 2, after the complete bipartite graph $G(C, P, W)$ is constructed, we eliminate all the edges with weight $W(Tai, Tbj) > \omega$ and then execute the Hungarian algorithm to find $gmin^P$ with the minimum mapping similarity distance, $MinMSD^P($ $La, Lb)$. Then, we compute the position distance $PD^P(La,$ $Lb, fmin^P)$ using Equations 8.24 or 8.25. Finally, XLS-PP is computed using Equation 8.26 and 8.27 for Spearman's footrule and Kendall tau respectively.

Total Complexity of the algorithm is $O(k^2 \cdot Cost(TS(Ta_i,$ $Tb_j)) + k^3 + k^2)$. Note the additional $O(k^2)$ to compute the position component. The complexity of this algorithm is same as the one for XLS-P.

8.7 Evaluation of Top-k XML Lists

Most of the related work was presented in Section 8.2.

XML Retrieval Evaluation: The INitiative for the Evaluation of XML Retrieval (INEX) [INi09] has provided since 2002 the infrastructure and means for evaluating the effectiveness of content-oriented XML search systems. INEX utilizes a series of queries that may contain both content and structural conditions. Although XML retrieval allows document fragments to be retrieved, these fragments cannot always be viewed as independent units. In this direction, INEX is encouraging the development of systems that return entities instead of just documents or elements. Our work can benefit this initiative of INEX by providing appropriate evaluation measures for lists of XML fragments. Clarke [Cla05] and Kazai et al. [KLdV04] present techniques to incorporate the overlap between XML fragments when evaluating XML search algorithms. They are complementary to our work since their techniques can be applied on our measures to account for overlap between the XML results.

Matching in Relational Databases: Guha et al. [Guh04] address the problem of merging approximate attribute rankings produced by executing a query on a "dirty" relational database. To do so, they propose a modification to the Hungarian Algorithm to identify a set of top ranking results.

In our case, the top-k lists are fairly small and hence memory-based matching techniques like the Hungarian algorithm are more appropriate.

8.8 Experimental Evaluation

In this section we experimentally evaluate the measures presented in the previous sections by comparing three popular XML keyword search algorithms. We use tree edit distance (TED) as the XML tree similarity measure (TS).

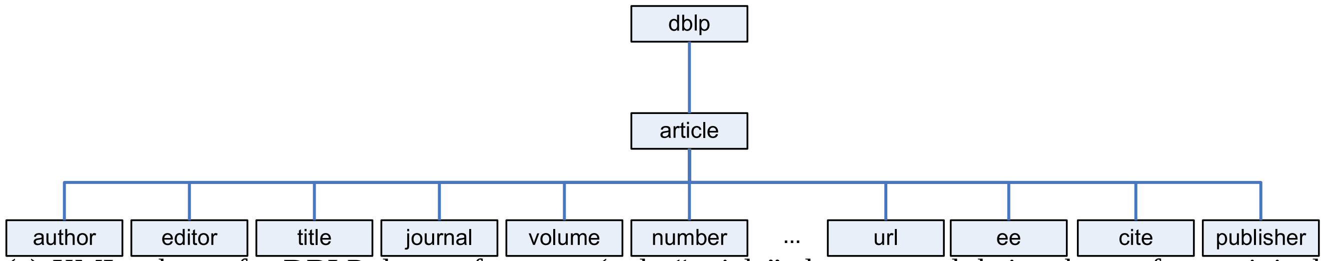
8.8.1 Datasets and Experimental Setup

Datasets: We use two real datasets: the DBLP dataset and the NASA XML dataset available at [oWCSE09]. Figure 8.2 shows a reduced version of both datasets' schemata and Table 8.2 summarizes their characteristics.

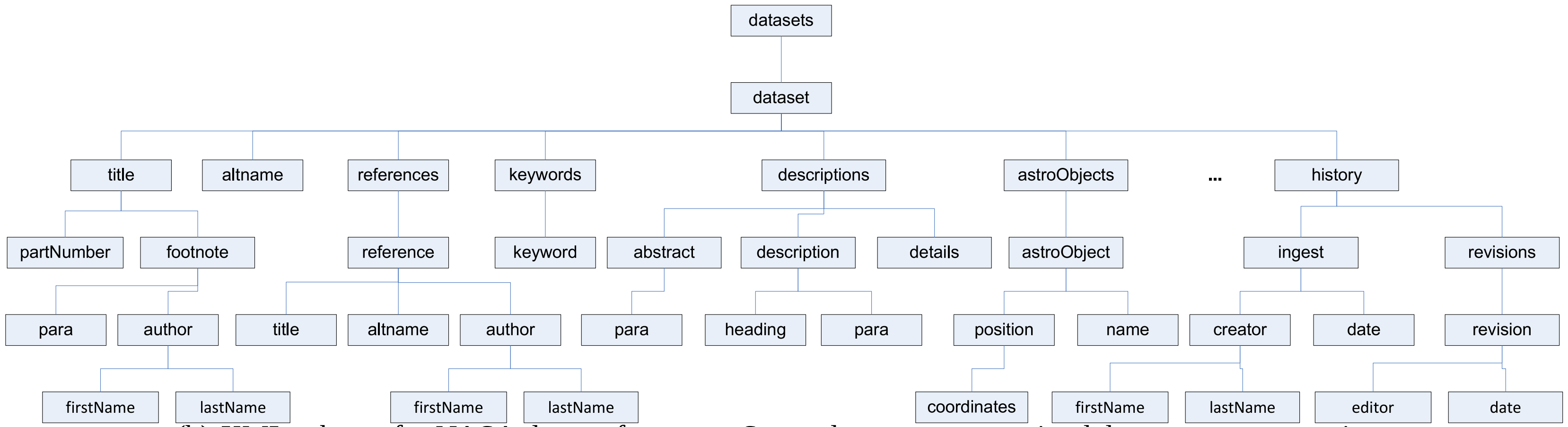
Table 8.2: XML Datasets Used in the Experiments.

Dataset	Number of Elements	Average Depth	Maximum Depth
DBLP	7137933	1.90	5
NASA	791923	5.58	8

Experimental Setup: We implemented the following XML keyword proximity search systems: XRANK [GSBS03], XSEarch [CMKS03] and XKeyword [HPB03]. These three algorithms take as input a corpus of XML documents and a keyword query, and return as output an ordered list of XML fragments that satisfy the query by containing all the keywords. All three algorithms favor minimal and compact subtrees that satisfy the query, but use different ranking functions and pruning rules. In particular, while XKeyword ranks its answers by the size of the resulting subtree, XRANK and XSEARCH also utilize Information Retrieval (IR) score functions based on $tf \cdot idf$. XSEarch prunes result paths that repeat the same tag in internal nodes, while XRANK prunes



(a) XML schema for DBLP dataset fragment (only "article" elements and their subtrees from original dataset are included)



(b) XML schema for NASA dataset fragment. Some elements were omitted due to space constraints.

Figure 8.2: XML schemata for DBLP and NASA datasets.

results if there is a more specific result in the same element. Also, XRANK returns whole subtrees while XSEarch and XKeyword return paths.

In our implementation, we used the IR score provided by the CONTAINSTABLE function of Microsoft SQL Server 2000 to compute the IR components of both XRANK and XSEARCH ranking functions. The experiments were performed on a PC with an Intel Pentium Core 2 Duo, 2.00 GHz processor, 2GB RAM, running Windows Vista Business. All algorithms were developed in Java (JDK version 1.6.0_06), use the Document Object Model (DOM) for XML parsing and navigation, and Microsoft SQLServer 2000 for the persistent storage of indexes. The tree similarity (TS) measure we use in our experiments is the dynamic programming algorithm by Zhang and Shasha [ZS89] which computes the tree-edit-distance between ordered trees [Bil05] whose complexity is
$Cost(TED(Tai, Tbj)) = O(|Ta_i||Tb_j| \cdot min(leaves(Ta_i), depth(Ta_i)) \cdot min(leaves(Tb_j), depth(Tb_j))$.
We refer to a detailed survey of tree edit distance algorithms [Bil05]. In Section 8.8.2, we first analyze the results of a single query to show the intuition of our evaluation scheme, and later we report average XML Lists Distance values over many experiments on the two datasets. In Section 8.8.3, we report performance (time) experimental results.Figure 3 shows a reduced version of both datasets' schemata and Table 8.2 summarizes their characteristics.

8.8.2 Quantitative Results

Analyze a Single Query: To illustrate our measures, we present an analysis for the keyword query "*database retrieval language*" over the DBLP XML dataset. Figure 8.3 shows the top-3 search results output by each of the three XML search algorithms. Table 8.3 presents the XLS, XLS-P and XLS-PP measures between every pair of XML lists from Figure 8.3. Notice that $XLS(La, Lb)$ is in [0,1] while XLS-$P(La, Lb)$ and XLS-$P(La, Lb)$ are in [0,2] and we found that setting the distance measure constants to $a = 1$ and $b = 1$ leads to reasonable results.

Notice that in Table 8.3 we only present XLS-PP measures for $\omega = 0.7$ and 0.9 as XLS-PP values for $\omega = 0.5$, 0.3, 0.1 for the top-3 lists presented in Figure 8.3 are the same as for $\omega = 0.7$

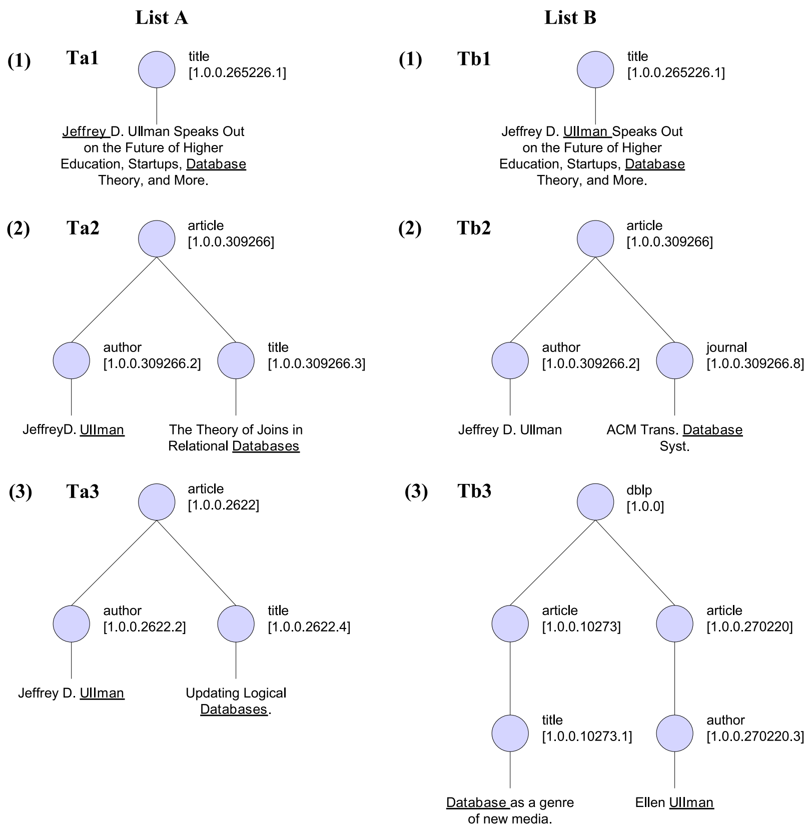


Figure 8.3: Top-3 search results for query *“database retrieval language”* over DBLP.

Table 8.3: XML List distances based on Total and Partial mappings for top-k lists in Figure 8.3.

$XLS[MinMSD^T(La, Lb)]$ $XLS\text{-}P[XLS\text{-}P^F, XLS\text{-}P^K]$ $XLS\text{-}PP[XLS\text{-}PP^F, XLS\text{-}PP^K]$	**XRANK**	**XSEARCH**	**XKEYWORD**
XRANK	$XLS[0.00]$ $XLS\text{-}P[0.00, 0.00]$ $XLS\text{-}PP^{0.7}[0.00, 0.00]$	$XLS[0.30]$ $XLS\text{-}P[0.80, 0.30]$ $XLS\text{-}PP^{0.7}[0.67, 0.50]$	$XLS[0.30]$ $XLS\text{-}P[1.05, 0.46]$ $XLS\text{-}PP^{0.7}[0.75, 0.54]$
XSEARCH	$XLS[0.30]$ $XLS\text{-}P[0.80, 0.30]$ $XLS\text{-}PP^{0.7}[0.67, 0.50]$ $XLS\text{-}PP^{0.9}[0.49, 0.41]$	$XLS[0.00]$ $XLS\text{-}P[0.00, 0.00]$ $XLS\text{-}PP^{0.7}[0.00, 0.00]$ $XLS\text{-}PP^{0.9}[0.00, 0.00]$	$XLS[0.00]$ $XLS\text{-}P[0.25, 0.17]$ $XLS\text{-}PP^{0.7}[0.08, 0.04]$ $XLS\text{-}PP^{0.9}[0.08, 0.04]$
XKEYWORD	$XLS[0.30]$ $XLS\text{-}P[1.05, 0.46]$ $XLS\text{-}PP^{0.7}[0.75, 0.54]$ $XLS\text{-}PP^{0.9}[0.58, 0.45]$	$XLS[0.00]$ $XLS\text{-}P[0.25, 0.17]$ $XLS\text{-}PP^{0.7}[0.08, 0.04]$ $XLS\text{-}PP^{0.9}[0.08, 0.04]$	$XLS[0.00]$ $XLS\text{-}P[0.00, 0.00]$ $XLS\text{-}PP^{0.7}[0.00, 0.00]$ $XLS\text{-}PP^{0.9}[0.00, 0.00]$

(we explain why later). Note that we use the penalty constant c equal to ω.

Let La, Lb and Lc be the top-3 lists of XRANK, XSEarch and XKeyword algorithms respectively as shown in Figure 8.3. The associated tree edit distance values between every pair of XML trees in each of the lists as follows:

$$AB = \begin{array}{c|ccc|} & Tb_1 & Tb_2 & Tb_3 \\ Ta_1 & 0.00 & 0.50 & 0.71 \\ Ta_2 & 0.98 & 0.98 & 0.89 \\ Ta_3 & 0.50 & 0.00 & 0.71 \end{array} \quad AC = \begin{array}{c|ccc|} & Tb_1 & Tb_2 & Tb_3 \\ Ta_1 & 0.00 & 0.50 & 0.71 \\ Ta_2 & 0.98 & 0.98 & 0.89 \\ Ta_3 & 0.50 & 0.00 & 0.71 \end{array}$$

$$BC = \begin{array}{c|ccc|} & Tb_1 & Tb_2 & Tb_3 \\ Ta_1 & 0.00 & 0.50 & 0.71 \\ Ta_2 & 0.50 & 0.00 & 0.71 \\ Ta_3 & 0.71 & 0.71 & 0.00 \end{array}$$

First of all, notice that the top-3 lists of XSEarch and XKeyword are identical (and hence the tree edit distance matrices AB and AC are identical), except that the first two results of XKeyword have the same score. This is the reason that the distances between XSEarch and XKeyword, for total mapping, are small (but not zero) in Table 8.3, since the position components consider ties. Note that XRANK returns a different subtree as its second result, since the XRANK function ranks the total score for this subtree higher than the score of the single element that appears in the other two lists. In this subtree, the keyword "*Retrieval*"appears twice within the "*title*" element, which increases its IR score. In addition, the third element in the XRANK list was penalized by its length and as a result.

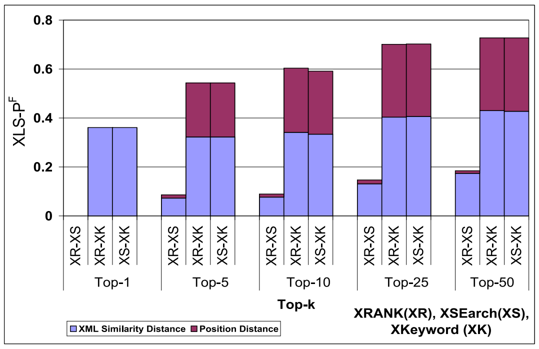
Between XRANK and XSEarch, two results are identical, and Ta_2 is mapped to Tb_3 in total mapping, even though they are very different. This irrelevant mapping and is removed in the partial mapping measures.

We computed $XLS\text{-}PP$ for various thresholds, $\omega = 0.9$, 0.7, 0.5, 0.3 and 0.1, and found that $XLS\text{-}PP$ distance values are identical for thresholds 0.7, 0.5, 0.3 and 0.1.This is because we have at least two identical trees between every pair of list (three identical trees in case of XSEarch and XKeyword) and they always get mapped between them, while the third unmapped result get mapped with the unmapped result in the other list depending on the threshold. Between XRANK & XSEarch, and XRANK & XKeyword, the tree edit distance of this third mapping is 0.89 and hence the results are identical for thresholds 0.1, 0.3, 0.5, 0.7.

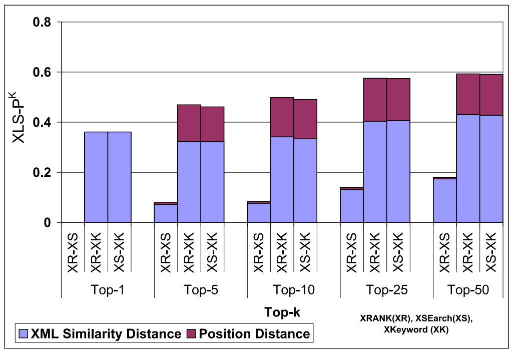
Quantitative Results over Multiple Queries

Figures 8.4(a) and 8.4(b) show the total distances (split into the two components) between the result lists pro-duced by the three search algorithms on the DBLP dataset averaged over 50 two-keyword queries, using $XLS\text{-}P^F$ and $XLS\text{-}P^K$, respectively. The queries used include: *"artificial intelligence", "xml indexing", "text mining", "image retrieval", "OLAP mining"*. Notice that the distance increases as k increases because as the trees get larger, the results become more disparate due to the pruning rules of the algorithms that go in effect for larger trees. As mentioned before, XKeyword ranks its answers by the size of the resulting subtree, while XRANK and XSEARCH also utilize Information Retrieval (IR) score functions based on $tf \cdot idf$. The reason that XKeyword has large distance to the other two rankings is that it does not have an IR component in its ranking function. Hence, when multiple trees have the same size, they are ranked arbitrarily. XRANK and XSEarch have smaller distance between them because their rankings are more similar given that the results were mostly single-node trees.

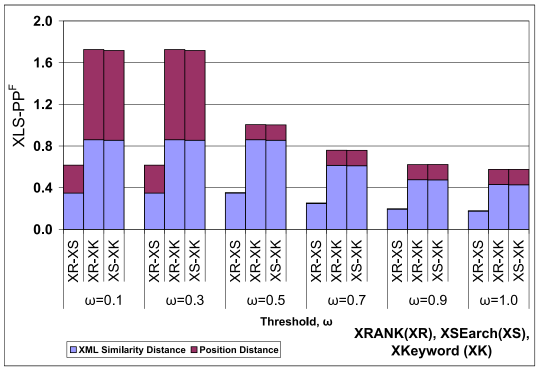
Figures 8.4(c) and 8.4(d) show the distances between the results of the three search algorithms, for the DBLP dataset, averaged over 50 two-keyword queries using $XLS\text{-}P^F$ and $XLS\text{-}P^K$, for varying thresholds, for top-50 results respectively. Recall that for $\omega = 1.0$, $XLS\text{-}PP$ (partial mapping) reduces to $XLS\text{-}P$ (total mapping). We use the penalty constant c equal to ω. In Figures 6(c), 6(d), 6(e) and 6(f), we see that the normalized distances increase as decreases. The reason is that for small there are few matches which lead to large position distance components. Note that for XRANK-vs.-XKeyword and XSEarch-vs.-XKeyword, for $\omega = 0.9$ we get slightly smaller distances than total mapping ($\omega = 1.0$). The reason is that almost all tree pairs in the top-50 results of these rankings have normalized tree edit distance up to 0.7, while for $\omega = 1.0$, we divide by a larger number (than for $\omega = 0.9$) to normalize the XML similarity component. On the other hand, for XRANK-vs.-XSEarch, the distance keeps reducing as ω increases from 0.1 to 0.9 and this is because there are some tree pairs in the top-50 results of these rankings with



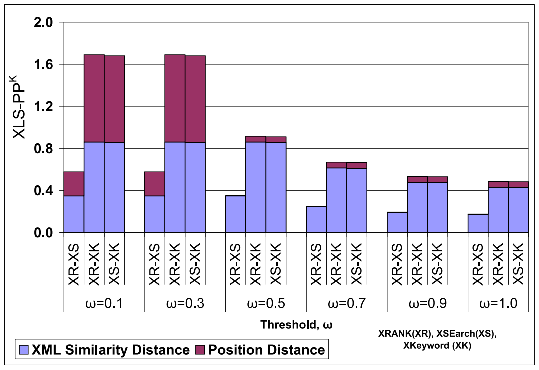
(a) Average $XLS\text{-}P^F$ vs. Top-k



(b) Average $XLS\text{-}P^K$ vs. Top-k



(c) Average $XLS\text{-}PP^F$ vs. threshold, ω, $k = 50$



(d) Average $XLS\text{-}PP^K$ vs. threshold, ω, $k = 50$

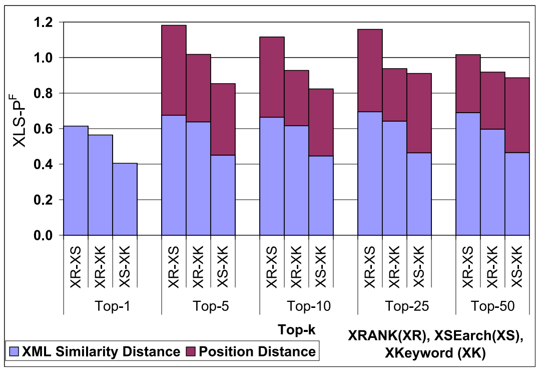
Figure 8.4: Experiments on DBLP Dataset.

normalized tree edit distance greater than 0.9.

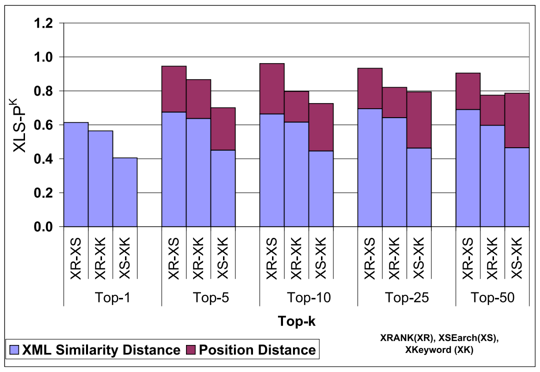
Figure 8.5 repeats the set of experiments of Figure 8.4 on the NASA dataset. Some sample two-keyword queries used in these experiments are: *"arcminutes magnitude", "astrographic motion", "equinox culmination", "photo-graphic wavelengths", "oxford zone"*. Some important observations on the results of NASA dataset are (a) Distance between XML lists is generally larger for NASA dataset because of its larger depth. (b) In contrast to Figure 8.2, XSEarch and XKeyword have the smallest distance because both algorithms return paths as result. This factor was less important in Figure 8.2 because most results were single-node. In contrast, XRANK has large distance to the other two rankings because it returns whole subtree as result. (c) XRANK is very close to XSEarch in DBLP, but very far in NASA dataset. The reason is that the XRANK and XSEarch pruning conditions are very rare for very shallow subtrees (DBLP) but more frequent for deeper subtrees (NASA dataset). The latter also leads to unpredictable fluctuations to the distances for increasing k (Figure 8.5), in contrast to the linear increase in the DBLP dataset (Figure 8.4). In both datasets, notice that the XML Similarity distance contributes the most to the total distance. This shows that the main difference of these three algorithms comes more from how they define a result and less on how they rank them.

8.8.3 Performance Results

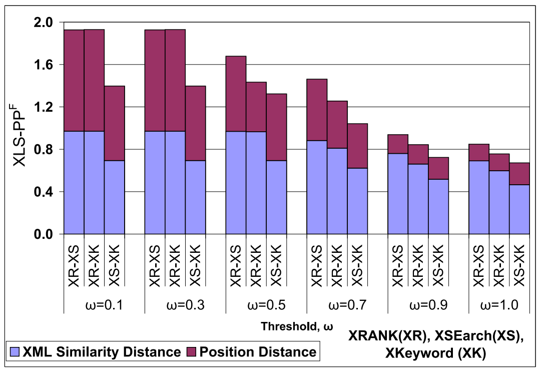
Due to space constraints and negligible execution times for the DBLP dataset (always less than one second), we only present results on the deeper NASA dataset. Figure 8.6(a) shows the average execution time to compute XLS-P for various values of k, over the same 50 two-keyword queries used in the distance experiments. As expected, the average execution time increases superlinearly as k increases because there are more results in the top-k lists under comparison. Figure 8.6(b) shows the average execution time to compute XLS-PP for various values of the threshold ω, for fixed $k = 50$. Notice that the execution times are different for the three pairs of search algorithms. The reason is that XRANK produces the largest size of results as it returns whole XML elements, while XKeyword produces concise results by returning paths. XSEarch produces results of intermediate size by returning paths like XKeyword but has different pruning rules.



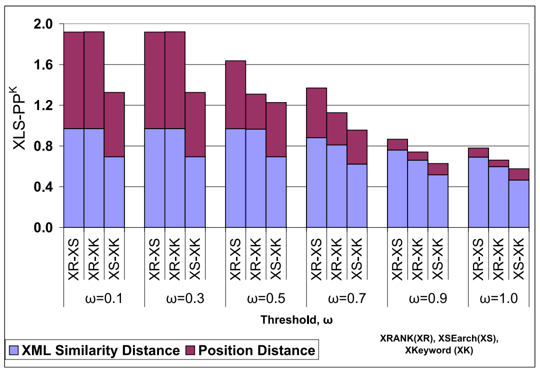
(a) Average $XLS\text{-}P^F$ vs. Top-k



(b) Average $XLS\text{-}P^K$ vs. Top-k



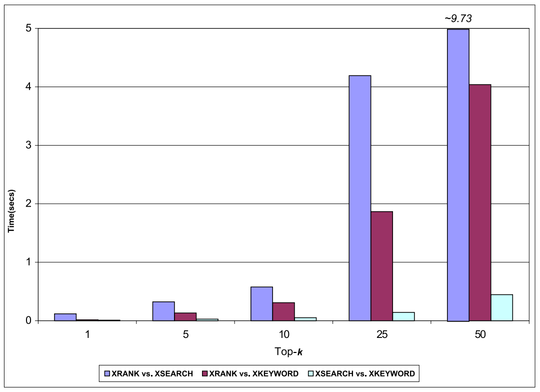
(c) Average $XLS\text{-}PP^F$ vs. threshold, ω, $k = 50$



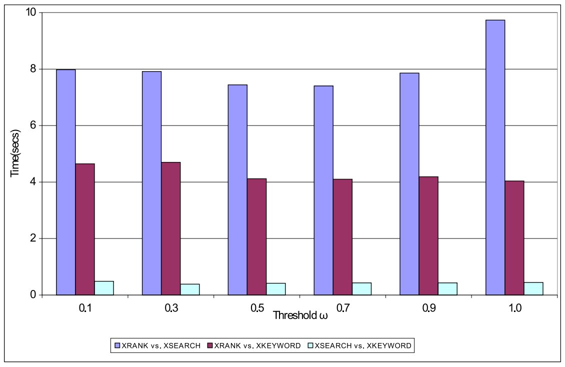
(d) Average $XLS\text{-}PP^K$ vs. threshold, ω, $k = 50$

Figure 8.5: Experiments on NASA Dataset.

Thus, the execution times of XRANK vs. XSEarch are the highest, while XSEarch vs. XKeyword is the lowest.



(a) Avg. execution time to compute $XLS\text{-}P$ ($\omega = 1.0$) vs. Top-k



(b) Avg. execution time to compute $XLS\text{-}PP$ ($\omega = 1.0$) vs. Top-k

Figure 8.6: Performance Experiments on NASA Dataset.

CHAPTER 9

CONCLUSIONS

In this dissertation I have explored three different challenges that must be addressed to facilitate and enhance the massive adoption of semistructured documents and, in particular, of the Extensible Markup Language. These three challenges have been clearly identified as *Storage*, *Parsing* and domain-specific *Information Discovery* on such type of documents. Each of these challenges has been deeply explored and novel solutions have been proposed to improve the performance and quality of each if these aspects.

A novel method for *storing* semistructured documents has been proposed, mapping the physical characteristics of semistructured documents to the geometrical layout of hard drives. Such optimization facilitates navigation of the data by reducing access overheads, and is achieved by utilizing information provided by standard disk profiling tools.

To provide an optimal *parsing* and processing of semistructured documents, we have developed a Double-Lazy Parser, a new approach that responds to the need of a more memory-efficient XML DOM parser, by introducing lazy behavior in both the pre-parsing and progressive parsing phases.

Extending the previous work on searching semistructured documents, we have created a framework that exploits the domain-specific knowledge to improve the quality of the *information discovery* process. In particular, we have created the XOntoRank system, that integrates the domain knowledge captured by clinical ontologies into a system for searching Electronic Health Records.

To evaluate the results of our search system for semistructured documents, we designed meaningful evaluation metrics that deal with top-k lists of subtrees instead of objects, taking into consideration the tree similarity and the position distance among the lists.

BIBLIOGRAPHY

[AAB+05] S. Abiteboul, R. Agrawal, P. Bernstein, M. Carey, S. Ceri, B. Croft, D. DeWitt, M. Franklin, H. García-Molina, D. Gawlick, J. Gray, L. Haas, A. Halevy, J. Hellerstein, Y. Ioannidis, M. Kersten, M. Pazzani, M. Lesk, D. Maier, J. Naughton, H. Schek, T. Sellis, A. Silberschatz, M. Stonebraker, R. Snodgrass, J. Ullman, G. Weikum, J. Widom, and S. Zdonik. The Lowell database research self-assessment. *Commun. ACM*, 48(5):111–118, 2005.

[Abr90] S. Abramsky. The Lazy Lambda Calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Welsey, Reading, MA, 1990.

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: A system for keyword-based search over relational databases. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 627–627, New York, NY, USA, 2002. ACM.

[AGM+90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[AM06] Loredana Afanasiev and Maarten Marx. An analysis of the current xquery benchmarks. In *ExpDB*, pages 9–20, 2006.

[AMM05] L. Afanasiev, I. Manolescu, and P. Michiels. MemBeR: A Micro-benchmark Repository for XQuery. In Stéphane Bressan, Stefano Ceri, Ela Hunt, Zachary G. Ives, Zohra Bellahsene, Michael Rys, and Rainer Unland, editors, *Database and XML Technologies, Third International XML Database Symposium, XSym 2005, Trondheim, Norway, August 28-29, 2005, Proceedings*, volume 3671 of *Lecture Notes in Computer Science*, pages 144–161. Springer, 2005.

[And00] Rie Kubota Ando. Latent semantic space: iterative scaling improves precision of inter-document similarity measurement. In *SIGIR '00: Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 216–223, New York, NY, USA, 2000. ACM.

[AST07] ASTM Continuity of Care Record (CCR).
http://www.centerforhit.org/x201.xml, 2007.

[AYBS04] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: a full-text search extension to XQuery. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 583–594, New York, NY, USA, 2004. ACM.

[AYCD06] Sihem Amer-Yahia, Emiran Curtmola, and Alin Deutsch. Flexible and efficient XML search with complex full-text predicates. In *SIGMOD '06: Proceedings of* *the 2006 ACM SIGMOD international conference on Management of data*, pages 575–586, New York, NY, USA, 2006. ACM.

[Baa03] Franz Baader. *The Description Logic Handbook : Theory, Implementation and Applications*. Cambridge University Press, January 2003.

[BBM$^+$01] Denilson Barbosa, Attila Barta, Alberto O. Mendelzon, George A. Mihaila, Flavio Rizzolo, and Patricia Rodriguez-Guianolli. Tox - the toronto XML engine. In *Workshop on Information Integration on the Web*, pages 66–73, 2001.

[BCJ$^+$05] Kevin Beyer, Roberta J. Cochrane, Vanja Josifovski, Jim Kleewein, George Lapis, Guy Lohman, Bob Lyle, Fatma Ozcan, Hamid Pirahesh, Normen Seemann, Tuong Truong, Bert Van der Linden, Brian Vickery, and Chun Zhang. System rx: One part relational, one part xml. In *SIGMOD*, 2005.

[BD03] Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall, Englewood Cliffs, NJ, second edition, September 2003.

[BDL$^+$] Stephane Bressan, Gillian Dobbie, Zoe Lacroix, Mong Li Lee, Ying Guang Li, Ullas Nambiar, and Bimlesh Wadhwa. XOO7: Applying OO7 benchmark to XML query processing tool. pages 167–174.

[BFHR06] Medha Bhadkamkar, Fernando Farfan, Vagelis Hristidis, and Raju Rangaswami. Efficient Native Storage for Semi-structured Data (extended paper version). In *http://www.cis.fiu.edu/SSS/NativeXMLextended.pdf*, 2006.

[BFRS02a] Phil Bohannon, Juliana Freire, Prasan Roy, and Jérôme Siméon. From XML schema to relations: A cost-based approach to XML storage. In *In ICDE*, pages 64–75, 2002.

[BFRS02b] Philip Bohannon, Juliana Freire, Prasan Roy, and Jérôme Siméon. From XML Schema to Relations: A Cost-based Approach to XML Storage. *ICDE*, 2002.

[BGBJ05] Rafae Bhatti, Arif Ghafoor, Elisa Bertino, and James B. D. Joshi. X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. *ACM Trans. Inf. Syst. Secur.*, 8(2):187–227, 2005.

[BGC03] John Bucy, Gregory Ganger, and Contributors. The DiskSim Simulation Environment Version 3.0 Reference Manual. *Carnegie Mellon University Technical Report CMU-CS-03-102*, January 2003.

[BH06] Srikanta Bedathur and Jayant Haritsa. Search-optimized suffix-tree storage for biological applications. In David A. Bader, Manish Parashar, Sridhar Varadarajan, and Viktor K. Prasanna, editors, *12th IEEE International Conference on High Performance Computing (HiPC)*, volume 3769 of *Lecture Notes in Computer Science*, pages 29–39, Goa, India, October 2006. IEEE, Springer.

[BHP04] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, pages 564–575, 2004.

[Bil03] Philip Bille. Tree edit distance, alignment distance and inclusion. Technical report, IT-Universitetet i København, 2003.

[Bil05] Philip Bille. A survey on tree edit distance and related problems. *Theor. Comput. Sci*, 337:217–239, 2005.

[BL08] T. Bernes-Lee. Universal Resources Identifiers. http://www.w3.org/designissues/axioms.html, 2008.

[BLM05] Giorgio Busatto, Markus Lohrey, and Sebastian Maneth. Efficient memory representation of xml documents. In *DBPL*, pages 199–216, 2005.

[BLS06] F. Baader, C. Lutz, and B. Suntisrivaraporn. Efficient Reasoning in $\mathcal{EL}^+$. In *Proceedings of the 2006 International Workshop on Description Logics (DL2006)*, CEUR-WS, 2006.

[BNH$^+$02] Gaurav Bhalotia, Charuta Nakhe, Arvind Hulgeri, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in databases using BANKS. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 431–440, Washington, DC, USA, 2002. IEEE Computer Society.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[BPSM$^+$06] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan. Extensible Markup Language (XML) 1.1. W3C Recommendation, World Wide Web Consortium. http://www.w3.org/TR/xml11/, 2006.

[BR01] Timo Böhme and Erhard Rahm. Xmach-1: A benchmark for xml data management. In *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), 9. GI-Fachtagung,*, pages 264–273, London, UK, 2001. Springer-Verlag.

[BR03] Timo Böhme and Erhard Rahm. Multi-user evaluation of xml data management systems with xmach-1. In *Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers*, pages 148–158, London, UK, 2003. Springer-Verlag.

[But04] David Buttler. A short survey of document structure similarity algorithms. In Hamid R. Arabnia and Olaf Droegehorn, editors, *International Conference on Internet Computing*, pages 3–9. CSREA Press, 2004.

[BYRN99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Cal08] California Clinical Data Project: Setting Standards. http://www.chcf.org/documents/CCDPProjectOverview.pdf, 2008.

[CDA07] CDA. HL7 Clinical Document Architecture, Release 2.0. In *http://lists.hl7.org/read/attachment/61225/1/CDA-doc 20version.pdf. 2007*, 2007.

[CDF+94] M. Carey, D. DeWitt, M. Franklin, N. Hall, M. McAuliffe, J. Naughton, D. Schuh, M. Solomon, C. K. Tan, O. Tsatalos, S. White, and M. Zwilling. Shoring up Persistent Applications. In *ACM SIGMOD*, 1994.

[CGRM06] Venkatesan T. Chakaravarthy, Himanshu Gupta, Prasan Roy, and Mukesh Mohania. Efficiently linking text documents with relevant structured information. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 667–678. VLDB Endowment, 2006.

[CKKS05] Sara Cohen, Yaron Kanza, Benny Kimelfeld, and Yehoshua Sagiv. Interconnection semantics for keyword search in XML. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 389–396, New York, NY, USA, 2005. ACM.

[Cla05] Charles L. A. Clarke. Controlling overlap in content-oriented XML retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 314–321, New York, NY, USA, 2005. ACM.

[CMKS03] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSEarch: a semantic search engine for XML. In *VLDB'2003: Proceedings of the 29th international conference on Very large data bases*, pages 45–56. VLDB Endowment, 2003.

[CMM+03] David Carmel, Yoelle S. Maarek, Matan Mandelbrod, Yosi Mass, and Aya Soffer. Searching XML documents via XML fragments. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 151–158, New York, NY, USA, 2003. ACM.

[Cre97] F. Crestani. Application of Spreading Activation Techniques in InformationRetrieval. *Artif. Intell. Rev.*, 11(6):453–482, 1997.

[CS05] Werner Ceusters and Barry Smith. Tracking referents in electronic health records. *Stud Health Technol Inform*, 116:71–76, 2005.

[CSF03] Werner Ceusters, Barry Smith, and Jim Flanagan. Ontology and medical terminology: why description logics are not enough. In *Proceedings of TEPR 2003-towards an electronic patient record*, 2003.

[DAB$^+$06] Dolin, Alschuler, Boyer, Beebe, Behlen, Biron, and Shabo Shvo. HL7 Clinical Document Architecture Release 2. *J Am Med Inform Assoc.*, 13(1), Jan-Feb 2006.

[DAYF] Fang Du, Sihem Amer-Yahia, and Juliana Freire. ShreX: Managing XML Documents in Relational Databases.

[DCjWS06] Theodore Dalamagas, Tao Cheng, Klaas jan Winkel, and Timos Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31:187–228, 2006.

[DCWS04] Theodore Dalamagas, Tao Cheng, Klaas-Jan Winkel, and Timos K. Sellis. Clustering XML Documents by Structure. In *SETN*, pages 112–121, 2004.

[DFS99] Alin Deutsch, Mary F. Fernandez, and Dan Suciu. Storing Semistructured Data with STORED. *ACM SIGMOD*, 1999.

[DHM05] Xin Dong, Alon Y. Halevy, and Jayant Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, pages 85–96, 2005.

[Dia88] P. Diaconis. Group representation in probability and statistics. *IMS Lecture Notes Monograph Series*, 1988.

[DKF$^+$99] A.L. Delcher, S. Kasif, R.D. Fleischmann, J. Peterson, O. White, and S.L. Salzberg. Alignment of Whole Genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[DKNS01] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web, 2001.

[DMDH02] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 662–673, New York, NY, USA, 2002. ACM.

[DOM08] Document Object Model (DOM). http://www.w3.org/DOM/, 2008.

[DR03] Zoran Dimitrijevic and Raju Rangaswami. Quality of Service Support for Real-time Storage Systems. *Proceedings of International IPSI Conference*, October 2003.

[DRC$^+$04] Zoran Dimitrijevic, Raju Rangaswami, Edward Chang, David Watson, and Anurag Acharya. Diskbench: User-level Disk Feature Extraction Tool. *UCSB Technical Report TR-2004-18*, 2004.

[DRR08] O'Neil Delpratt, Rajeev Raman, and Naila Rahman. Engineering succinct dom. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 49–60, New York, NY, USA, 2008. ACM.

[EBB$^+$05] Peter L. Elkin, Steven H. Brown, Brent A. Bauer, Casey S. Husser, William Carruth, Larry R. Bergstrom, and Dietlind L. Wahner-Roedler. A controlled trial of automated classification of negation from clinical notes. *BMC Medical Informatics and Decision Making*, 5(13), May 2005.

[Fel98] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, May 1998.

[FG01] Norbert Fuhr and Kai Großjohann. XIRQL: a query language for information retrieval in XML documents. In *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 172–180, New York, NY, USA, 2001. ACM.

[FGKL02] Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas. INEX: INitiative for the Evaluation of XML Retrieval. In Ricardo Baeza-Yates, Norbert Fuhr, and Yoelle S. Maarek, editors, *Proceedings of the SIGIR 2002 Workshop on XML and Information Retrieval*, 2002.

[FJS96] Adam Finkelstein, Charles E. Jacobs, and David H. Salesin. Multiresolution video. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 281–290, New York, NY, USA, 1996. ACM.

[FKM$^+$04] Ronald Fagin, Ravi Kumar, Mohammad Mahdian, D. Sivakumar, and Erik Vee. Comparing and aggregating rankings with ties. In *In PODS*, pages 47–58, 2004.

[FKS03] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 28–36, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[FLMM06] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and searching xml data via two zips. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 751–760, New York, NY, USA, 2006. ACM.

[Flo08] Florida International University School of Computing and Information Sciences. XOntoRank Project Homepage.
http://dsrl.cs.fiu.edu/projects/ir_biomedical/xontorank_homepage/, 2008.

[FMMP05] Sergio Flesca, Giuseppe Manco, Elio Masciari, and Luigi Pontieri. Fast Detection of XML Structural Similarity. *IEEE Trans. on Knowl. and Data Eng.*, 17(2):160–175, 2005. Student Member-Pugliese,, Andrea.

[FMPP02] Sergio Flesca, Elio Masciari, Luigi Pontieri, and Andrea Pugliese. Detecting Structural Similarities Between XML Documents. In *In Proc. of the 5th Intl. Workshop on the Web and Databases*, pages 55–60, 2002.

[Fra04] Massimo Franceschet. XPathMark: An XPath Benchmark For XMark. *University of Amsterdam Technical Report PP-2004-04*, 2004.

[Gal07] Galax. Galax. *http://www.galaxquery.org*, 2007.

[Gan01] Gregory R. Ganger. Blurring the Line Between OSes and Storage Devices. *Carnegie Mellon University Technical Report CMU-CS-01-166*, December 2001.

[GKP02] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *VLDB*, 2002.

[GML08] GML. Geography markup language. *http://opengis.net/gml/*, 2008.

[GMOS02] Todd J. Green, Gerome Miklau, Makoto Onizuka, and Dan Suciu. Processing XML Streams with Deterministic Automata. In *ICDT '03: Proceedings of the 9th International Conference on Database Theory*, pages 173–189, London, UK, 2002. Springer-Verlag.

[GNA$^+$] Garth A. Gibson, Dave F. Nagle, Khalil Amiri, Jeff Butler, Fay W. Chang, Howard Gobioff, Charles Hardin, Erik Riedel, David Rochberg, and Jim Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. *Proceedings of the ACM ASPLOS*, Oct 98.

[Gro08] Object Management Group. UML Resource Page. http://www.uml.org/, 2008.

[Gru02] Torsten Grust. Accelerating XPath Location Steps. In *ACM SIGMOD*, 2002.

[GSBS03] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 16–27, New York, NY, USA, 2003. ACM.

[Guh04] Sudipto Guha. Merging the results of approximate match operations. In *In VLDB*, pages 636–647, 2004.

[GVD05] Fatih Gelgi, Srinivas Vadrevu, and Hasan Davulcu. Improving Web Data Annotations with Spreading Activation. In *WISE*, pages 95–106, 2005.

[GW97] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *VLDB*, 1997.

[HCL05] Chia-Hsin Huang, Tyng-Ruey Chuang, and Hahn-Ming Lee. Prefiltering techniques for efficient XML document processing. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 149–158, New York, NY, USA, 2005. ACM.

[HCLL06] Chia-Hsin Huang, Tyng-Ruey Chuang, James J. Lu, and Hahn-Ming Lee. XML Evolution: a two-phase XML processing model using XML prefiltering techniques. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 1215–1218. VLDB Endowment, 2006.

[Hea08] Health Insurance Portability and Accountability Act.
http://www.hipaa.org/, 2008.

[Hel07] Sven Helmer. Measuring the structural similarity of semistructured documents using entropy. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 1022–1032. VLDB Endowment, 2007.

[Her02] William R Hersh. Medical informatics: improving health care through information. *JAMA*, 288(16):1955–1958, 2002.

[HGP03] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient IR-style keyword search over relational databases. In *VLDB'2003: Proceedings of the 29th international conference on Very large data bases*, pages 850–861. VLDB Endowment, 2003.

[HHP06] Heasoo Hwang, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: a system for authority-based search on databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 796–798, New York, NY, USA, 2006. ACM.

[Hit09] Hitachi. Ultrastar 10K300 Product Summary.
http://www.hitachigst.com/tech/techlib.nsf/techdocs/
619188AFEBC5E88486256E43006FFFBF/$file/10K300_PS.pdf, 2009.

[HL707a] HL7 Reference Information Model Billboard.
http://www.miforum.net/distillate/rim/Graphics/RI, 2007.

[HL707b] HL7 V3.0 Data Types Specification.
http://aurora.regenstrief.org/v3dt/report.html, 2007.

[HL708a] HL7. Health level seven xml. *http://www.hl7.org/special/Committees/xml/xml.htm*, 2008.

[HL708b] HL7 Reference Information Model.
http://www.hl7.org/library/datamodel/RIM/C30204/rim.htm, 2008.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *VLDB '02: Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681. VLDB Endowment, 2002.

[HP06] Vagelis Hristidis and Yannis Papakonstantinou. Keyword proximity search in XML trees. *IEEE Transactions on Knowledge and Data Engineering*, 18(4):525–539, 2006. Member-Nick Koudas and Member-Divesh Srivastava.

[HPB03] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword Proximity Search on XML Graphs. *ICDE*, 2003.

[HRS99] U. Hahn, M. Romacker, and S. Schulz. How knowledge drives understanding–matching medical ontologies with the needs of medical language processing. *Artif Intell Med*, 15(1):25–51, January 1999.

[HS95] Roger L. Haskin and Frank L. Stein. A system for the delivery of interactive television programming. In *Compcon: Digest of papers*, pages 209–215, 1995.

[HSW$^+$04] Larry Huston, Rahul Sukthankar, Rajiv Wickremesinghe, M. Satyanarayanan, Gregory R. Ganger, Erik Riedel, and Anastassia Ailamaki. Diamond: A Storage Architecture for Early Discard in Interactive Search. *Proceedings of the USENIX Conference on File and Storage Technologies*, March 2004.

[ID01] Sitaram Iyer and Peter Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous i/o. In *Symposium on Operating Systems Principles*, pages 117–130, 2001.

[INi09] INitiative for the Evaluation of XML Retrieval. http://inex.is.informatik.uni-duisburg.de/2007/, 2009.

[IV02] T. Itälä and A. Virtanen. Seamless Care and CDA: Finland (Aluetietojaerjestelmae). In *HL7 International CDA Conference*, 2002.

[JAKC$^+$02] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. TIMBER: A Native XML Database. *The VLDB Journal*, 11(4):274–291, 2002.

[JWZ94] Tao Jiang, Lusheng Wang, and Kaizhong Zhang. Alignment of trees - an alternative to tree edit. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 75–86, London, UK, 1994. Springer-Verlag.

[KBM05] Carl-Christian Kanne, Matthias Brantner, and Guido Moerkotte. Cost-Sensitive Reordering of Navigational Primitives. *SIGMOD*, 2005.

[KBNK02] Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. Covering Indexes for Branching Path Queries. *SIGMOD*, 2002.

[KC06] Jong Wook Kim and K. Sel#231;uk Candan. CP/CV: concept similarity mining without frequency information from domain describing taxonomies. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 483–492, New York, NY, USA, 2006. ACM.

[KG90] Maurice Kendall and Jean D. Gibbons. *Rank Correlation Methods*. A Charles Griffin Title, 5 edition, September 1990.

[KH05] Manaka Kenji and Sato Hiroyuki. Static optimization of XSLT stylesheets: template instantiation optimization and lazy XML parsing. In *DocEng '05: Proceedings of the 2005 ACM symposium on Document engineering*, pages 55–57, New York, NY, USA, 2005. ACM.

[Kis01] Oleg Kiselyov. A Better XML Parser through Functional Programming. *Lecture Notes in Computer Science*, 2257:209+, 2001.

[KK05] Myung Sook Kim and Yong-Hae Kong. Ontology-DTD Matching Algorithm for Efficient XML Query. In *FSKD (2)*, pages 1093–1102, 2005.

[KKJ06] Myung Sook Kim, Yong-Hae Kong, and Chang Wan Jeon. Remote-Specific XML Query Mobile Agents. In *DEECS*, pages 143–151, 2006.

[KLdV04] Gabriella Kazai, Mounia Lalmas, and Arjen P. de Vries. The overlap problem in content-oriented XML retrieval evaluation. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 72–79, New York, NY, USA, 2004. ACM.

[KM77] Sukhamay Kundu and Jaydev Misra. A Linear Tree Partition Algorithm. In *SIAM J. Comput.*, pages 6(1):151–154, March 1977.

[KM99] C. Kanne and G. Moerkotte. Efficient Storage of XML Data . *Universitaet Mannheim Technical Report*, 1999.

[KM06] Carl-Christian Kanne and Guido Moerkotte. A linear time algorithm for optimal tree sibling partitioning and approximation algorithms in Natix. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 91–102. VLDB Endowment, 2006.

[KOG$^+$01] R. Kukafka, P. W. O'Carroll, J. L. Gerberding, E. H. Shortliffe, C. Aliferis, J. R. Lumpkin, and W. A. Yasnoff. Issues and opportunities in public health informatics: a panel discussion. *J Public Health Manag Pract*, 7(6):31–42, November 2001.

[KPH98] Kimberly Keeton, David A. Patterson, and Joseph M. Hellerstein. A Case for Intelligent Disks (IDISKS). *SIGMOD Record*, 27(3):42–52, September 1998.

[LAE$^+$04] Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 108–119. VLDB Endowment, 2004.

[LCP06] Wei Lu, Kenneth Chiu, and Yinfei Pan. A Parallel Approach to XML Parsing. *The 7th IEEE/ACM International Conference on Grid Computing (Grid2006)*, Barcelona, Spain, September 28-29, 2006.

[LCS$^+$04] Wang Lian, David W. Cheung, Ieee Computer Society, Nikos Mamoulis, and Siu ming Yiu. An efficient and scalable algorithm for clustering XML documents by structure. *IEEE Transactions on Knowledge and Data Engineering*, 16:82–96, 2004.

[Les69] M. E. Lesk. Word-word association in document retrieval systems. *American Documentation*, 20(1):27–38, 1969.

[LH03] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology, 2003.

[Lin98] Dekang Lin. An Information-Theoretic Definition of Similarity. In *ICML*, pages 296–304, 1998.

[LM01] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. *VLDB Journal*, 2001.

[Log06] Logical Observation Identifiers Names and Codes (LOINC). http://www.regenstrief.org/medinformatics/loinc/, 2006.

[LS00] Hartmut Liefke and Dan Suciu. XMill: an efficient compressor for XML data. pages 153–164, 2000.

[LYJ04] Yunyao Li, Cong Yu, and H. V. Jagadish. Schema-free XQuery. In *VLDB'2004: Proceedings of the Thirtieth international conference on Very large data bases*, pages 72–83. VLDB Endowment, 2004.

[MAG$^+$97] Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, and Jennifer Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, 1997.

[Max09] Maxtor. DiamondMaxPlus D740X Fact Sheet. http://www.seagate.com/staticfiles/maxtor/en_us/documentation/data_sheets/diamondmax_D740X_datasheet.pdf, 2009.

[MH04] Sergio L. S. Mergen and Carlos A. Heuser. Matching of XML Schemas and Relational Schemas. In *SBBD*, 2004.

[MHS$^+$03] Clement J. McDonald, Stanley M. Huff, Jeffrey G. Suico, Gilbert Hill, Dennis Leavelle, Raymond Aller, Arden Forrey, Kathy Mercer, Georges DeMoor, John Hook, Warren Williams, James Case, and Pat Maloney. LOINC, a Universal Standard for Identifying Laboratory Observations: A 5-Year Update. *Clin Chem*, 49(4):624–633, 2003.

[Mit97] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[ML95] Joseph F. McCarthy and Wendy G. Lehnert. Using decision trees for coreference resolution. In *IJCAI*, pages 1050–1055, 1995.

[MLLA03] Xiaofeng Meng, Daofeng Luo, Mong-Li Lee, and Jing An. Orientstore: A schema based native xml storage system. In *VLDB*, pages 1057–1060, 2003.

[MML08] MML. Medical Markup Language. *http://www.ncbi.nlm.nih.gov/*, 2008.

[MMM06] Ioana Manolescu, Cédric Miachon, and Philippe Michiels. Towards micro-benchmarking xquery. In *ExpDB*, pages 28–39, 2006.

[MMRV05] Ana G. Maguitman, Filippo Menczer, Heather Roinestad, and Alessandro Vespignani. Algorithmic detection of semantic similarity. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 107–116, New York, NY, USA, 2005. ACM.

[MNJ04] P. Mitra, N. F. Noy, and A. R. Jaiswal. OMEN: A Probabilistic Ontology Mapping Tool. In *Workshop on Meaning coordination and negotiation at the Third International Conference on the Semantic Web (ISWC-2004)*, 2004.

[MNU00] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178, New York, NY, USA, 2000. ACM.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[MW03] Andrew McCallum and Ben Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In Subbarao Kambhampati and Craig A. Knoblock, editors, *IIWeb*, pages 79–84, 2003.

[Nat06] Natix. http://www.dataexmachina.de/. 2006.

[NC01] Vincent Ng and Claire Cardie. Improving machine learning approaches to coreference resolution. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 104–111, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[NJ02] Andrew Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, pages 61–66, Madison, Wisconsin, USA, June 2002.

[NJ03] Matthias Nicola and Jasmi John. Xml parsing: A threat to database performance. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 175–178, 2003.

[NKS07] Matthias Nicola, Irina Kogan, and Berni Schiefer. An xml transaction processing benchmark. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 937–948, New York, NY, USA, 2007. ACM.

[NLB$^+$01] U. Nambiar, Z. Lacroix, S. Bressan, M. Lee, and Y. Li. Xml benchmarks put to the test, 2001.

[NLM08] NLM Unified Medical Language System.
http://www.nlm.nih.gov/research/umls/, 2008.

[NM03] Natalya F. Noy and Mark A. Musen. The PROMPT suite: interactive tools for ontology merging and mapping. *Int. J. Hum.-Comput. Stud.*, 59(6):983–1024, 2003.

[NNP00] Igor Nekrestyanov, Boris Novikov, and Ekaterina Pavlova. An analysis of alternative methods for storing semistructured data in relations. In *ADBIS-DASFAA*, pages 354–361, 2000.

[NSL02] Markus L. Noga, Steffen Schott, and Welf Löwe. Lazy XML processing. In *DocEng '02: Proceedings of the 2002 ACM symposium on Document engineering*, pages 88–94, New York, NY, USA, 2002. ACM.

[OBJ03] Adeniyi Onabajo, Iryna Bilykh, and Jens Jahnke. Wrapping legacy medical systems for integrated health network. In *Migration and Evolvability of Long-life Software Systems (MELLS-03) Workshop at the Conference NetObjectDays 2003*, 2003.

[ODS08] ODS. Open Document Specification v1.0. *http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf*, 2008.

[Ont07] Ontology matching. http://www.ontologymatching.org/, 2007.

[oox08] OpenOffice XML File Format v1.0. http://xml.openoffice.org/xml_specification.pdf, 2008.

[ope08] OpenDocument Specification v1.0. http://www.oasis-open.org/committees/download.php/12572/OpenDocument-v1.0-os.pdf, 2008.

[oWCSE09] University of Washington Computer Science and Engineering. XML Data Repository. http://www.cs.washington.edu/research/xmldatasets/ www/repository.html, 2009.

[PB99] H. A. Proper and P. D. Bruza. What is information discovery about? *J. Am. Soc. Inf. Sci.*, 50(9):737–750, 1999.

[PGMW95] Yannis Papakonstantinou, Hector Garcia-Molina, and Jennifer Widom. Object Exchange Across Heterogeneous Information Sources. In *ICDE '95: Proceedings of the Eleventh International Conference on Data Engineering*, 1995.

[PHOE02] S. L. Price, W. R. Hersh, D. D. Olson, and P. J. Embi. SmartQuery: context-sensitive links to medical knowledge sources from the electronic patient record. *Proc AMIA Symp*, pages 627–631, 2002.

[PIC07] PICNIC, Professionals and Citizens Network for Integrated Care. http://www.medcom1-4.dk/picnic/default.htm, 2007.

[Pra06] Sujeet Pradhan. An algebraic query model for effective and efficient retrieval of XML fragments. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 295–306. VLDB Endowment, 2006.

[Qua09] Quantum. Fireball Plus KX Fact Sheet. http://www.seagate.com/staticfiles/maxtor/en_us/documentation/ quantum_jumper_settings/fireball_plus_kx_ata_jumpers.pdf, 2009.

[Res99] Philip Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.

[RFHR] M. Ramanath, J. Freire, J. Haritsa, and P. Roy. Searching for efficient XML to relational mappings.

[RGF98] Erik Riedel, Garth Gibson, and Christos Faloutsos. Active Storage For Large-Scale Data Mining and Multimedia. *Proceedings of the VLDB*, August 1998.

[RMBB89] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, Jan/Feb 1989.

[Rok07] Daniel Rokhsar. Computational Analysis of Genomic Sequence Data. 2007.

[RPJ$^+$03] K. Runapongsa, J. Patel, H. Jagadish, Y. Chen, and S. Al-Khalifa. The michigan benchmark: Towards xml query performance diagnostics, 2003.

[RW94a] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[RW94b] Christ Ruemmler and John Wilkes. An Introduction to Disk Drive Modeling. *Computer*, 2:17–28, 1994.

[RxN07] RxNorm. United States National Library of Medicine. http://www.nlm.nih.gov/research/umls/rxnorm/index.html, 2007.

[Sal89] Gerard Salton. *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[SAL$^+$07] Saurav Sahay, Eugene Agichtein, Baoli Li, Ernest V. Garcia, and Ashwin Ram. Semantic annotation and inference for medical knowledge discovery. NSF Next Generation Data Mining(NGDM) Symposium, 2007.

[sax08] Simple API for XML (SAX). http://www.saxproject.org/, 2008.

[SB97] Gerard Salton and Chris Buckley. Improving retrieval performance by relevance feedback. *Readings in Information Retrieval*, pages 355–364, 1997.

[SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive deduplication using active learning. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–278, New York, NY, USA, 2002. ACM.

[SCI07] SCIPHOX Stenogramm. http://www.sciphox.de/, 2007.

[Sea09] Seagate. Cheetah 15K.4 Product Overview. http://www.seagate.com/content/docs/pdf/marketing/Seagate_Cheetah_15K-4.pdf, 2009.

[SGG06] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts 7th Edition with Java 7th Edition*. John Wiley & Sons, 2006.

[Sin01] Amit Singhal. Modern Information Retrieval: A Brief Overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4):35–42, 2001.

[SN03] Steffen Schott and Markus L. Noga. Lazy XSL transformations. In *DocEng '03: Proceedings of the 2003 ACM symposium on Document engineering*, pages 9–18, New York, NY, USA, 2003. ACM.

[SNO08] SNOMED Clinical Terms (SNOMED CT). http://www.snomed.org/snomedct/index.html, 2008.

[Spa97] Spackman, KA and Campbell, KE and CÃ, RA and others. SNOMED-RT: a reference terminology for health care. In *Proceedings of the 1997 AMIA Annual Fall Symposium*, pages 640–644, 1997.

[SPP$^+$03] Muthian Sivathanu, Vijayan Prabhakaran, Florentina I. Popovici, Timothy E. Denehy, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. Semantically-Smart Disk Systems. *Proceedings of the USENIX Symposium on File and Storage Technologies*, pages 73–88, March 2003.

[SSP$^+$05] Steven W. Schlosser, Jiri Schindler, Stratos Papadomanolakis, Minglong Shao, Anastassia Ailamaki, Christos Faloutsos, and Gregory R. Ganger. On Multidimensional Data and Modern Disks. *Proceedings of the 4th USENIX Conference on File and Storage Technology*, December 2005.

[SSS$^+$04] Jiri Schindler, Steven W. Schlosser, Minglong Shao, Anastassia Ailamaki, and Gregory R. Ganger. Atropos: A Disk Array Volume Manager for Orchestrated Use of Disks. *Proceedings of the USENIX Conference on File and Storage Technologies*, March 2004.

[STW03] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Ontology-Enabled XML Search. In *Intelligent Search on XML Data*, pages 119–131, 2003.

[STW05] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Semantic Similarity Search on Semistructured Data with the XXL Search Engine. *Inf. Retr.*, 8(4):521–545, 2005.

[STZ$^+$] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB 1999*.

[SVG08] SVG. Scalable vector graphics. *http://www.w3.org/Graphics/SVG/*, 2008.

[SWK$^+$02a] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management, 2002.

[SWK$^+$02b] Albrecht Schmidt, Florian Waas, Martin L. Kersten, Michael J. Carey, Ioana Manolescu, and Ralph Busse. XMark: A Benchmark for XML Data Management. *VLDB*, 2002.

[TADP99] Nisha Talagala, Remzi H. Arpaci-Dusseau, and David Patterson. Microbenchmark-based Extraction of Local and Global Disk Characteristics. *UC Berkeley Technical Report*, 1999.

[Tai79] Kuo-Chung Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[Tex07] Text REtrieval Conference (TREC). http://trec.nist.gov/, 2007.

[TH02] Pankaj Tolani and Jayant R. Haritsa. XGRIND: A query-friendly XML compressor. In *ICDE*, 2002.

[The03] Anja Theobald. An Ontology for Domain-oriented Semantic Similarity Search on XML Data. In *BTW*, pages 217–226, 2003.

[TKM02] Sheila Tejada, Craig A. Knoblock, and Steven Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 350–359, New York, NY, USA, 2002. ACM.

[TW02a] Anja Theobald and Gerhard Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *In EDBT*, pages 477–495, 2002.

[TW02b] Anja Theobald and Gerhard Weikum. The XXL search engine: ranked retrieval of XML data using indexes and ontologies. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 615–615, New York, NY, USA, 2002. ACM.

[UML07] UMLS Metathesaurus Fact Sheet. http://www.nlm.nih.gov/pubs/factsheets/umlsmeta.html, 2007.

[Uni07] Unified modeling language. http://www.uml.org/, 2007.

[vLEB$^+$04] J. van Lunteren, T. Engbersen, J. Bostian, B. Carey, and C. Larsson. XML Accelerator Engine. *First International Workshop on High Performance XML Processing*, 2004.

[WD01] P. Mitra G. Wiederhold and S. Decker. A scalable framework for interoperation of information sources. In *The 1st International Semantic Web Working Symposium (SWWS01)*, 2001.

[WGPW95] B. Worthington, G. Ganger, Y. Patt, and J. Wilkes. Online Extraction of SCSI Disk Drive Parameters. *Proceedings of ACM Sigmetrics Conference*, pages 146–156, 1995.

[Win95] W. E. Winkler. The state of record linkage and current research problems. *Statistics in Medicine*, 14:491–498, 1995.

[WLS07] Raymond K. Wong, Franky Lam, and William M. Shui. Querying and maintaining a compact xml storage. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1073–1082, New York, NY, USA, 2007. ACM.

[WN05] Melanie Weis and Felix Naumann. DogmatiX tracks down duplicates in XML. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 431–442, New York, NY, USA, 2005. ACM.

[WR05] D. Wollersheim and W. J. Rahayu. Using Medical Test Collection Relevance Judgements to Identify Ontological Relationships Useful for Query Expansion. In *ICDEW*

'05: Proceedings of the 21st International Conference on Data Engineering Workshops, page 1160, Washington, DC, USA, 2005. IEEE Computer Society.

[Xal07] Xalan. Xalan-Java. *http://xml.apache.org/xalan-j*, 2007.

[XC96] Jinxi Xu and W. Bruce Croft. Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 4–11, New York, NY, USA, 1996. ACM.

[xer08] Apache Xerces2 Java Parser. http://xml.apache.org/xerces2-j/, 2008.

[xin08] XML Inclusion. http://www.w3.org/TR/xinclude/, 2008.

[XLWS06] Jianjun Xu, Jiaheng Lu, Wei Wang, and Baile Shi. Effective Keyword Search in XML Documents Based on MIU. In Mong-Li Lee, Kian-Lee Tan, and Vilas Wuwongse, editors, *DASFAA*, volume 3882 of *Lecture Notes in Computer Science*, pages 702–716. Springer, 2006.

[XP05] Yu Xu and Yannis Papakonstantinou. Efficient keyword search for smallest LCAs in XML databases. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 527–538, New York, NY, USA, 2005. ACM.

[XPa07] XPath. XML Path Language (XPath) Version 1.0.
http://www.w3.org/TR/xpath, 2007.

[xpo08] XML Pointer Language (XPointer) Version 1.0. http://www.w3.org/TR/WD-xptr, 2008.

[xpp08] XML Pull Parser (XPP). http://www.extreme.indiana.edu/xgws/xsoap/xpp/, 2008.

[xpu08] XML Pull Parsing. http://www.xmlpull.org/index.shtml, 2008.

[XQu07] XQuery 1.0 and XPath 2.0 Full Text.
http://www.w3.org/TR/xquery-full-text/, 2007.

[XSL07] XSLT. Xslt version 1.0. *http://www.w3.org/TR/xslt*, 2007.

[xsl08] Extensible Stylesheet Language (XSL). http://www.w3.org/TR/xsl/, 2008.

[XT07] XT. XT. *http://www.blnz.com/xt/index.html*, 2007.

[YKT05] Rui Yang, Panos Kalnis, and Anthony K. H. Tung. Similarity evaluation on tree-structured data. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 754–765, New York, NY, USA, 2005. ACM.

[YOK03] Benjamin B. Yao, M. Tamer Özsu, and John Keenleyside. Xbench - a family of benchmarks for xml dbmss. In *Proceedings of the VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web-Revised Papers*, pages 162–164, London, UK, 2003. Springer-Verlag.

[ZAR02] Dmitry Zelenko, Chinatsu Aone, and Anthony Richardella. Kernel methods for relation extraction. In *EMNLP '02: Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, pages 71–78, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[ZS89] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

APPENDIX

This Appendix presents a sample Electronic Medical Record using the Clinical Document Architecture (CDA) as described in Section 6.2.

Listing 9.1: HL7 CDA Sample Document

```
<? xml version="1.0" ?>
<ClinicalDocument xmlns="urn:hl7-org:v3" xmlns:voc="urn:hl7-org:v3/voc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="urn:hl7-org:v3 CDA.ReleaseTwo.Committee.2004.xsd" templateId="2.16.840.1.113883.3.27.1776">
 <id extension="c266" root="2.16.840.1.113883.3.933"/>
 <confidentialityCode code="N" codeSystem="2.16.840.1.11.3883.5.25" />
 <author>
  <time value="20040407"/>
  <assignedAuthor>
   <id extension="KP00017" root="2.16.840.1.113883.3.933"/>
   <assignedPerson>
    <name>
     <given>Juan</given>
     <family>Woodblack</family>
     <suffix>MD</suffix>
 </name></assignedPerson></assignedAuthor></author>
 <recordTarget>
  <patientRole>
   <id extension="49912" root="2.16.840.1.113883.3.933"/>
   <patientPatient>
    <name>
     <given>FirstName</given>
     <family>LastName</family>
     <suffix>Jr.</suffix>
    </name>
    <administrativeGenderCode code="M" codeSystem="2.16.840.1.5.1"/>
    <birthTime value="20020924"/>
   </patientPatient>
   <providerOrganization>
    <id extension="M345" root="2.16.840.1.113883.3.933"/>
 </providerOrganization></patientRole></recordTarget>
```

```
<component>
 <StructuredBody>
  <component>
   <section>
    <code code="10160-0" codeSystem="
       2.16.840.1.113883.6.1" codeSystemName="LOINC"/>
    <title>Medications</title>
    <entry>
     <Observation>
      <code code="84100007" codeSystem="
         2.16.840.1.113883.6.96" codeSystemName="SNOMED
         CT"  displayName="Medications"/>
      <value xsi:type="CD" code="195967001" codeSystem=
         "2.16.840.1.113883.6.96" codeSystemName="SNOMED
          CT" displayName="Asthma">
      <originalText><reference value="m1"/></
         originalText>
    </value></Observation></entry>
    <entry>
     <Observation>
      <code code="84100007" codeSystem="
         2.16.840.1.113883.6.96" codeSystemName="SNOMED
         CT" displayName="Medications"/>
      <value xsi:type="CD" code="32398004" codeSystem="
         2.16.840.1.113883.6.96" codeSystemName="SNOMED
         CT" displayName="Bronchitis">
       <value xsi:type="CD" code="91143003" codeSystem=
          "2.16.840.1.113883.6.96" codeSystemName="
          SNOMED CT" displayName="Albuterol" />
    </value></Observation></entry>
    <entry>
     <SubstanceAdministration>
      <text><content ID="m1">Theophylline</content>20
         mg every other day, alternating with 18 mg
         every other day. Stop if temperature is above
         103F.</text>
      <consumable>
       <manufacturedProduct>
        <manufacturedLabeledDrug>
         <code code="66493003" codeSystem="
            2.16.840.1.113883.6.96" codeSystemName="
            SNOMED CT"  displayName="Theophylline"/>
      </manufacturedLabeledDrug></manufacturedProduct></
         consumable>
    </SubstanceAdministration></entry>
  </section></component>
  <component>
   <section>
    <code code="11384-5" codeSystem="
       2.16.840.1.113883.6.1" codeSystemName="LOINC"/>
    <title>Physical Examination</title>
    <component>
     <section>
      <code code="8716-3" codeSystem="
         2.16.840.1.113883.6.1" codeSystemName="LOINC"/>
      <title>Vital Signs</title>
      <text>
       <table>
        <tr>
         <th>Temperature</th>
         <td>36.9 C 98.5 F</td>
        </tr>
        <tr>
         <th>Pulse</th>
         <td>86 / minute </td>
      </tr></table></text>
      <entry>
       <Observation>
        <code code="50373000" codeSystem="
           2.16.840.1.113883.6.96" codeSystemName="
           SNOMED CT" displayName="Body height"/>
        <effectiveTime value="200404071430"/>
        <value xsi:type="PQ" value="1.77" unit="m" />
    </Observation></entry></section></component></
       section></component>
</StructuredBody></component></ClinicalDocument>
```

VITA

FERNANDO FARFAN

February 12, 1978	Born, Guatemala City, Guatemala
1998–1999	Junior Programmer Grupo Polaris, S. A. Guatemala City, Guatemala
2001	B.S., Computer Science Universidad del Valle de Guatemala Guatemala City, Guatemala
2000–2004	Senior Analyst Banco de Desarrollo Rural Guatemala City, Guatemala
2005	M.S., Computer Science Florida International University Miami, Florida

PUBLICATIONS

F. Farfán and V. Hristidis. *Overview of XML*. Book Details: Information Discovery on Electronic Health Records. To be published by Taylor & Francis group in 2009. Editor: V. Hristidis.

F. Farfán, R. Varadarajan and V. Hristidis. *Electronic Health Records*. Book Details: Information Discovery on Electronic Health Records. To be published by Taylor & Francis group in 2009. Editor: V. Hristidis.

V. Hristidis, R. Varadarajan, F. Farfán. *Searching Electronic Health Records*. Book Details: Information Discovery on Electronic Health Records. To be published by Taylor & Francis group in 2009. Editor: V. Hristidis.

M. Bhadkamkar, F. Farfán, V. Hristidis and R. Rangaswami. *Storing Semi-structured Data on Disk Drives*. ACM Transactions on Storage Journal, Vol. 5, Issue # 2, May, 2009.

F. Farfán, V. Hristidis, A. Ranganathan, M. Weiner. *XOntoRank: Ontology-Aware Search on Electronic Medical Records*. Proceedings of IEEE ICDE 2009, Shanghai, China, March 2009.

F. Farfán, V. Hristidis and R. Rangaswami. *2LP: A double-lazy XML parser*. Elsevier Information Systems 34 (2009), pp. 145-163.

V. Hristidis, F. Farfán, R. P. Burke, A. F. Rossi and J. A. White. *Challenges for Information Discovery on Electronic Medical Records*. Book details: Next Generation of Data Mining. CRC -

Taylor & Francis. Editors: H. Kargupta, J. Han, P. Yu, R. Motwani, V. Kumar. 2008.

F. Farfán, V. Hristidis, A. Ranganathan and R. P. Burke. *Ontology-Aware Search on XML-based Electronic Medical Records*. Proceedings of IEEE ICDE 2008, Cancún, México, April 2008 (Poster paper).

V. Hristidis, F. Farfán, R. P. Burke, A. F. Rossi and J. A. White. *Information Discovery on Electronic Medical Records*. Proceedings of NSF NGDM 2007, Baltimore, Maryland. October 2007.

F. Farfán, V. Hristidis and R. Rangaswami. *Beyond Lazy XML Parsing*. Proceedings of DEXA 2007, Regensburg, Germany. September 2007.

M. Bhadkamkar, F. Farfán, V. Hristidis and R. Rangaswami. *Efficient Native Storage Systems for Semistructured Data*. Florida International University Technical Report FIU-SCIS-TR-2006-09-01. September 2006.

M. Bhadkamkar, F. Farfán, V. Hristidis and R. Rangaswami. *Storing Trees on Disk Drives*. Proceedings of USENIX FAST 2005, San Francisco, California (Short Paper).