Departament d'Enginyeria
i Ciència dels Computadors

Flexible techniques for heterogeneous XML data retrieval

Ph. D. Thesis

Presented by Ismael Sanz
Supervised by Dr. Rafael Berlanga Llavori and Dr. Marco Mesiti
Castelló, October 2007

Departament d'Enginyeria
i Ciència dels Computadors

Técnicas flexibles para la recuperación de datos XML heterogéneos

Ismael Sanz Blasco

Trabajo realizado bajo la dirección
del Dr. Rafael Berlanga Llavori y el Dr. Marco Mesiti
presentado en la Universitat Jaume I
para optar al grado de Doctor por la Universitat Jaume I

Castelló, Octubre de 2007

Als meus pares i al meu germà.

A Lledó.

Resumen extendido

Introducción

XML [Bray et al., 2006] se ha convertido en el estándar de hecho para representar una gran cantidad de información en todos los ámbitos. Esto ha motivado una intensa actividad investigadora, cuyo primer resultado práctico es que la mayor parte de sistemas de gestión de bases de datos actuales han ya incorporado facilidades para almacenar y consultar nativamente información en formato XML.

Tras la aparición de XML se desarrolló la distinción entre aplicaciones de XML que se han venido en llamar "centradas en los datos", en contraposición a los XML "centrados en los documentos". Los primeros se caracterizadas por (a) tener una estructura relativamente homogénea y (b) un contenido textual fuertemente tipado; la gran mayoría de estándares de comercio electrónico caen en esta categoría. En cambio, los segundos se caracterizan por incluir *contenido mixto*, como se denomina a las estructuras muy variables típicas de los documentos predominantemente textuales; el ejemplo paradigmático de aplicación XML centrada en los documentos es el lenguaje XHTML, utilizado para codificar las páginas web [W3C HTML Working Group, 2002].

Los estándares existentes para la gestión de información XML en bases de datos, y en particular los lenguajes XPath [Clark and DeRose, 1999] y XQuery [Boag et al., 2007], asumen que las colecciones sobre las que trabajan es "centradas en los datos", y los mecanismos que ofrecen están fuertemente orientados al recorrido de estructuras jerárquicas regulares. Un conjunto completamente distinto de técnicas se emplean para el manejo de de información textual, ejemplificado en los resultados presentados en las conferencias INEX (INitiative for the Evaluation of XML Retrieval). Como ejemplo híbrido cabe mencionar el sistema TeXQuery de [Amer-Yahia et al., 2004], una extensión de XQuery con primitivas de recuperación de la información.

Sin embargo, las aplicaciones caracterizadas como "centradas en datos" y "centradas en documentos" no representan más que una parte de las colecciones XML existentes. La adopción de XML como lenguaje para la representación y el intercambio de información por parte de diversas comunidades — bioinformática, patrimonio cultural, ontologías, sistemas de información geográfica y muchas otras — ha motivado la aparición de aplicaciones importantes para las que las técnicas desarrolladas hasta ahora no son eficaces, ya que ni se puede explotar una estructura relativamente regular ni la información puramente textual es predominante. Por ejemplo, en bioinformática hay una enorme cantidad de información etiquetada en XML, que abarca desde información molecular

a ontologías que describen los conceptos del dominio, pasando por el contenido de artículos técnicos. Según [MacMullen and Denn, 2005] muchas aplicaciones dependen de la utilización integrada de esas fuentes, pero en ese contexto las técnicas generales no son de aplicación directa, lo que motiva enormes esfuerzos en la creación de aplicaciones ad-hoc.

Esta tesis propone técnicas que permiten consultar colecciones XML que presentan un *grado muy alto de heterogeneidad*. Estas colecciones no pueden ser tratadas directamente con XPath y XQuery — ni tampoco con TeXQuery y técnicas similares — ya que no presentan heterogeneidad sólo a nivel textual, sino en todos los niveles posibles de granularidad que están presentes en cualquier colección de XML: texto, elementos, trayectorias, subárboles y documentos. Un objetivo adicional es que las técnicas propuestas sean *flexibles*, en el sentido de que sean lo suficientemente adaptables para minimizar el esfuerzo dedicado a desarrollar técnicas ad-hoc para cada caso en concreto.

Un punto crucial es que las técnicas desarrolladas deben permitir *consultas aproximadas*. Las características de las colecciones descritas impiden, como hemos visto, que se puedan utilizar técnicas de consulta "exactas", y es necesario desarrollar funciones de similaridad que permitan valorar la calidad de los resultados obtenidos por una consulta. A la hora de resolver un problema concreto que usa colecciones heterogéneas, la elección de una mediad de similaridad apropiada es un problema primario, y de hecho en la literatura hay descritas una gran cantidad de medidas sobre colecciones XML adaptadas al tratamiento de problemas concretos, junto con técnicas de consultas adaptadas exclusivamente para cada medida en particular. Sin embargo, muy pocas de esas aproximaciones se puede generalizar a problemas distintos, con lo que caemos una vez más en el problema de las soluciones ad-hoc. En consecuencia, en esta tesis también se desarrolla un *modelo genérico de medidas de similaridad* para XML, que permite utilizar las técnicas de consulta desarrolladas independientemente de la medida que se desee utilizar para resolver cada problema concreto.

Este último punto es particularmente importante si se tiene en cuenta que, incluso dentro de un mismo proyecto, a menudo diferentes usuarios requieren diferentes medidas de similaridad. Por ejemplo, un biólogo puede necesitar recuperar proteínas basándose en una comparación con una secuencia de aminoácidos, mientras que otro puede buscar "antígeno de la malaria" dentro de las descripciones textuales de la colección, y un tercero puede querer combinar ambos tipos de consulta. Esto nos lleva al concepto de *sistema orientado a la multi-similaridad* propuesto por [Adalı et al., 1998], que permite a sus usuarios elegir dinámicamente qué concepto de similaridad es el más apropiado en cada momento.

La solución propuesta en esta tesis para estos problemas está basada en la búsqueda de *fragmentos*. Los fragmentos son una manera concisa de expresar los requisitos de información del usuario que hace una consulta; consisten en una serie de elementos (etiquetas, atributos, texto, ...) con estructura. En su forma más simple se pueden considerar como patrones en forma de subárbol que se desea encontrar en la base de

datos, pero con una semántica muy flexible que, en gran parte, está *determinada por la medida de similaridad que se utilice*. Por ejemplo, se tendrá en cuenta si para calcular el resultado de la medida es relevante mantener el orden de padres e hijos o si no lo es, si se desea utilizar o no un tesauro para relajar el significado, o si al evaluación de algún componente del fragmento requiere utilizar alguna función específica. Potencialmente, cada medida impone restricciones diferentes, lo que hace necesario un modelo de medidas que permita procesarlas de forma coherente.

A continuación, trataremos la extensión de los algoritmos de consulta para soportar consultas de tipo top-k, que devuelven sólo los k resultados más relevantes. Este es un problema complejo ya que, definido genéricamente, el problema de recuperación de fragmentos se puede reducir a un problema de búsqueda no ordenada en árboles, que es NP-completo [Kilpeläinen, 1992]. La definición de fragmento introduce restricciones diseñadas para reducir la complejidad de los algoritmos a un coste lineal, pero al tratar grandes colecciones incluso este resultado puede ser problemático. Un segundo problema es que los principales algoritmos para hacer consultas top-k sobre XML están basados en el algoritmo propuesto por [Fagin et al., 2003], y por tanto requieren que la medida a tratar sea monótona, lo que no es deseable en nuestro contexto ya que perderíamos mucha flexibilidad. Hemos superado este problema usado técnicas basadas en las de [Xin et al., 2007] y que hasta ahora no se habían utilizado en el contexto de XML, consiguiendo significativas mejoras en la velocidad de consulta (entre el 250 % y el 350 % en los experimentos realizados).

Finalmente, presentaremos dos extensiones orientadas al postprocesamiento de la información obtenida mediante las técnicas de consulta que acabamos de esbozar. En primer lugar, trataremos la aplicación del modelo genérico de similaridad en algoritmos de agrupamiento, que aplicaremos para facilitar la interpretación de los resultados de las consultas aproximadas, que pueden llegar ser muy complejos. En segundo lugar, estudiaremos cómo modificar una implementación estándar de XQuery para que permita integrar las técnicas desarrolladas en la tesis en sistemas de bases de datos existentes.

La tesis se completa con una revisión de trabajos relevantes, y en particular de aproximaciones basadas en similaridad para XML; la descripción de un prototipo que implementa las técnicas descritas; y la presentación de varias líneas de investigación que pueden continuar el trabajo desarrollado aquí.

Metodología

Esta tesis está basada fundamentalmente en técnicas de procesamiento flexible de consultas para XML. En todos los algoritmos propuestos se realiza un estudio de su corrección y eficiencia, bien analítico o bien empírico, según sea más apropiado en cada caso. Los experimentos están basados en una combinación de colecciones reales y sintéticas.

El requisito fundamental del modelo flexible de medidas de similitud es la capacidad

de obtener medidas adaptadas a cada aplicación en particular mediante la combinación de medidas genéricas más sencillas, para lo que nos hemos basado en el patrón de diseño *composite*, un resultado básico en ingeniería del software orientado a objetos [Gamma et al., 1995]. La metainformación necesaria para describir las medidas está codificada usando lógicas descriptivas [Baader et al., 2003].

Aportaciones

- En esta tesis se define un marco general, basado en el concepto de *nivel de granularidad*, que permite clasificar las aplicaciones de XML basadas en similaridad. Este marco nos lleva a la definición de un concepto formal de *heterogeneidad*, basado en consideraciones de Teoría de la Información, para colecciones de documentos XML. Esto nos permite comparar y clasificar colecciones diversas según su grado de variabilidad.
- También se define un modelo general para la definición de medidas de similaridad basado en la composición de funciones básicas predefinidas. El modelo está orientado a sistemas basados en multi-similaridad, permitiendo crear o adaptar rápidamente medidas a aplicaciones específicas. La descripción de medidas así obtenidas mediante metainformación definida mediante lógica descriptiva facilita enormemente la creación de herramientas de apoyo para el desarrollo de medidas, y en particular el desarrollo colaborativo.
- La contribución central de la tesis son los algoritmos para la consulta flexible de colecciones XML definidos a través de fragmentos. Estos algoritmos, y las técnicas asociadas, permiten trabajar con colecciones que, pero su gran heterogeneidad, están fuera del alcance de las aproximaciones existentes, como XQuery, TeXQuery, y los diversos sistemas académicos para la recuperación aproximada de XML.
- A continuación se propone una versión top-k de los algoritmos anteriores. La principal novedad con respecto a los algoritmos existentes consiste en que, por primera vez, se hace posible utilizar medidas de similitud no monótonas en el contexto de XML.
- Para facilitar la aplicabilidad práctica de las técnicas arriba descritas, se proponen métodos para facilitar su incorporación de en sistemas de procesamiento XQuery: una especificación funcional que permite incorporar la nueva funcionalidad sin efectuar modificaciones en el lenguaje base, y un algoritmo de agrupamiento orientado al postprocesamiento de conjuntos de resultados particularmente complejos.
- Todas estas técnicas se han implementado en ArHeX, un conjunto de herramientas que ilustra la aplicación de la técnicas presentadas en esta tesis a la ingeniería de sistemas orientados a la multi-similaridad.

Trabajo futuro

La integración de las técnicas aquí presentadas en un sistema XML es una línea de investigación en la que se plantean varios problemas abiertos. Podemos citar los tres más obvios: la integración de los algoritmos aquí presentados como una opción más disponible para el optimizador, que podría acometerse siguiendo una aproximación similar a la de [Michiels et al., 2007]; la integración con las extensiones propuestas por TeXQuery, que proporcionan una manera natural de tratar consultas aproximadas y top-k, si bien orientadas a la recuperación de la información; y la posibilidad de extender las capacidades de XQuery para definir funciones, de tal manera que permiten definir nativamente medidas ajustadas al modelo propuesto en la tesis.

Otra contribución que abre posibilidades es la definición de un modelo formal para la heterogeneidad de las colecciones XML. En principio, esto hace posible definir técnicas (semi-)automáticas para determinar automáticamente qué medidas pueden ser más apropiadas para tratar colección, dados unos requisitos y la caracterización de la heterogeneidad de la colección obtenida automáticamente, en la línea de [Bernstein et al., 2005]. Esta aproximación podría facilitar enormemente el diseño de aplicaciones basadas en multi-similaridad.

El trabajo en el desarrollo de los prototipos ha mostrado que la capacidad de visualizar una colección XML heterogénea de manera global, dando la capacidad de determinar de un vistazo las distintas regiones de la colección en función de su características, resultaría extremadamente útil. Una aproximación que parece capaz de proporcionar una condensación de la información de esa magnitud puede ser el *paradigma de pixelización* [Lévy, 2007], posiblemente combinado con técnicas más convencionales.

Abstract

The progressive adoption of XML by new communities of users has motivated the appearance of applications that require the management of large and complex collections, which present a large amount of heterogeneity. Some relevant examples are present in the fields of bioinformatics, cultural heritage, ontology management and geographic information systems, where heterogeneity is not only reflected in the textual content of documents, but also in the presence of rich structures which cannot be properly accounted for using fixed schema definitions. Current approaches for dealing with heterogeneous XML data are, however, mainly focused at the content level, whereas at the structural level only a limited amount of heterogeneity is tolerated; for instance, weakening the parent-child relationship between nodes into the ancestor-descendant relationship.

The main objective of this thesis is devising new approaches for querying heterogeneous XML collections. This general objective has several implications: First, a collection can present different levels of heterogeneity in different granularity levels; this fact has a significant impact in the selection of specific approaches for handling, indexing and querying the collection. Therefore, several metrics are proposed for evaluating the level of heterogeneity at different levels, based on information-theoretical considerations. These metrics can be employed for characterizing collections, and clustering together those collections which present similar characteristics.

Second, the high structural variability implies that query techniques based on exact tree matching, such as the standard XPath and XQuery languages, are not suitable for heterogeneous XML collections. As a consequence, approximate querying techniques based on similarity measures must be adopted. Within the thesis, we present a formal framework for the creation of similarity measures which is based on a study of the literature that shows that most approaches for approximate XML retrieval (i) are highly tailored to very specific problems and (ii) use similarity measures for ranking that can be expressed as ad-hoc combinations of a set of "basic" measures. Some examples of these widely used measures are $tf \times idf$ for textual information and several variations of *edit distances*. Our approach wraps these basic measures into generic, parametrizable components that can be combined into complex measures by exploiting the *composite* pattern, commonly used in Software Engineering. This approach also allows us to integrate seamlessly highly specific measures, such as protein-oriented matching functions.

Finally, these measures are employed for the approximate retrieval of data in a context of highly structural heterogeneity, using a new approach based on the concepts of *pattern* and *fragment*. In our context, a pattern is a concise representations of the information

needs of a user, and a fragment is a match of a pattern found in the database. A pattern consists of a set of tree-structured elements — basically an XML subtree that is intended to be found in the database, but with a flexible semantics that is strongly dependent on a particular similarity measure. For example, depending on a particular measure, the particular hierarchy of elements, or the ordering of siblings, may or may not be deemed to be relevant when searching for occurrences in the database.

Fragment matching, as a query primitive, can deal with a much higher degree of flexibility than existing approaches. In this thesis we provide exhaustive and top-k query algorithms. In the latter case, we adopt an approach that does not require the similarity measure to be monotonic, as all previous XML top-k algorithms (usually based on Fagin's algorithm) do. We also presents two extensions which are important in practical settings: a specification for the integration of the aforementioned techniques into XQuery, and a clustering algorithm that is useful to manage complex result sets.

All of the algorithms have been implemented as part of ArHeX, a toolkit for the development of multi-similarity XML applications, which supports fragment-based queries through an extension of the XQuery language, and includes graphical tools for designing similarity measures and querying collections. We have used ArHeX to demonstrate the effectiveness of our approach using both synthetic and real data sets, in the context of a biomedical research project.

Keywords XML, heterogeneous data management, approximate query processing, similarity measures, ranked query results, top-k queries with non-monotonic functions, multi-similarity, clustering.

Acknowledgments

Many people have contributed to this thesis, and they have done it in many different ways. It is impossible in this brief note to show the extent of my gratitude to all of them.

First of all, I would like to thank my supervisors, Rafael Berlanga and Marco Mesiti. Without their support and advice, this work would have been impossible.

Rafa has been guiding me since the days in which, as an undergraduate at Universitat Jaume I, I approached him looking for a degree project in data management. When I came back to UJI after a hiatus of several years, and became a member of the Temporal Knowledge Bases Group (TKBG) and the Department of Computer Science and Engineering (DICC), his encouragement made me feel that I had never been away. I am grateful for his support and for the high standards he sets for himself and his collaborators. I feel lucky to belong to the group of wonderful people that is TKBG, and to enjoy the friendship of my current and former teammates: María José, Lola, Juanma, Ernesto, Victoria, Jordi, Roxana, Aurora and Henry. Some of them are now working with our industrial partner, Maat-GKnowledge, which also was my direct employer for a while. I appreciate their support and their willingness to do research. I am also thankful to the people of DICC for providing the framework for carrying on my research work.

I met Marco on a brief three-month stay at the Università degli Studi di Genova under a postgraduate Erasmus grant, which resulted in a fruitful collaboration and, more importantly, an enduring friendship. Most of the joint work with Marco, Giovanna Guerrini and Rafa, including many hours of discussions, has become a crucial part of this thesis. A big thanks to Marco and Giovanna, and all the great people I met in Genoa: Giorgio Delzanno (and little Alice), Barbara Catania, Anna Maddalena, Davide Ancona and everyone at DISI.

I cannot overstate how important as a formative experience was my three-year stay at the Vrije Universiteit Brussel from 1998 to 2001, at Prof. Robert Meersman's STARLab. I am proud of having met Miro Casanova, Mustafa Jarrar, Olga de Troyer, Peter Stuer, Robert, Sven Casteleyn, and many other great people. What I learned from them has been invaluable for me.

I would also like to thank all of the friends that set an example for me, or had a word of encouragement, or just been there when I needed them: Toni Morales, Gabriel Recatalá, Gus Casañ and Raül Marín, who are now colleagues at UJI; Micky González, Alex Ribes, Marisol García and Raül Romero, who pursued careers far away; Pilar, Paco, Marifé... An exceedingly long list of people should be included here, but it is impossible to do so. Thanks to you all.

Finally, I would also like to thank Alberto Abelló, Torben Bach Pedersen, Rafael Corchuelo, Michael Gould and Paolo Perlasca for accepting to be part of the evaluation process of this thesis, be it as external reviewers or as members of the jury.

Last, but not least, I want to thank Lledó for her patience and understanding, and also for being an example to me: her amazing ability to combine work ethic and joy of life has inspired and energized me to go through with this thesis. I'm happy to know that, just a few days after the public defense of this work, we will become husband and wife.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Preliminaries

Almost ten years after XML was standardized [Bray et al., 1998], the amount of information represented using it has grown dramatically, as it has been adopted by many communities as a natural way to encode and share content. As a consequence, the need for storing and querying large XML collections have grown in importance, and all major database management systems already support it natively. Unfortunately, supporting data management systems was not one of the design goals of XML: it was originally intended as a streamlined version of SGML [ISO 8879:1986, 1986], a hierarchical document markup language standardized by ISO that was deemed too complex for the requirements of the Web. Most XML-related terminology is derived directly from its SGML roots. XML *documents* consist of nested *elements*, which may possibly include *attributes* (name-value pairs). Elements may be empty, include some textual content (*PCDATA,* from "parsed character data") or contain nested elements.

The intended primary application of XML was to become the successor of HTML [W3C HTML Working Group, 2002], which used to be an application of SGML itself, as the language of choice for encoding documents available on the web. Very little input was provided by the database community. As a consequence, the original XML specification missed several crucial features that were required in order to create an efficient database system which could be compared to any relational DBMS. In particular, there was no suitable schema language, and no query language other than a programmatic interface, the DOM standard [Hors et al., 2004].

The original schema language for XML was the *Document Type Definition*, directly inherited from SGML. It included facilities for defining the hierarchical structure of elements in XML documents using a regular expression-like syntax, but did not contain a type system that could express constrains such as "the contents of the *currency* element must be *real numbers*". The content of elements could only be defined as *PCDATA*. This limitation was overcome by the specification of XML Schema [Fallside and Walmsley, 2004], which defined a proper type system for XML and made DTDs a relic. Still, the resulting data model presents some difficulties. Two are particularly relevant: (i) the tolerance of *optional elements*, or the fact that elements and whole structures may occur

an undetermined number of times, or completely absent and (ii) its *hierarchical* nature, which introduces a data model based on trees instead of tuples, thus making many natural operations on XML documents far more computationally costly than its relational analogues.

The development of XML query languages has required a large, and still ongoing, research effort. The first standard was XPath [Clark and DeRose, 1999], which provided a simple language for navigating XML documents using 14 "axes" (ancestor, descendant, ...) and returning the corresponding nodes. Even this simple language required new developments in indexing techniques for tree-like data in order to efficiently support all 14 axes, and the development of new join algorithms. This work provided a foundation for XQuery [Boag et al., 2007], the standard XML Query language, which builds a rich, Turing-complete functional language on top of XPath. It is an extraordinarily expressive and powerful query language, but it is also notoriously difficult to parse and optimize.

In the management of XML collections, a distinction was very soon established between "data-centric" and "document-centric" XML collections. The former are characterized by having a relatively homogeneous structure and strongly typed textual content, which makes them suitable for processing using XPath and XQuery; for instance, the great majority of standards for electronic commerce fall in this category. Instead, the latter are characterized by being predominantly textual documents, with some markup. These collections require a set of completely different techniques, adapted from plain text Information Retrieval to include some structural constraints, as exemplified in the results presented in the INEX (INitiative for the Evaluation of XML retrieval) series of conferences [Kazai et al., 2003]

1.2 Motivation and objectives

Thus, research in XML has focused around two poles, which are analogous to the distinction between data management and Information Retrieval systems. However, the applications typically characterized as data- and document-centric do not represent more than a part of existing XML collections. The adoption of XML as a language for the representation and exchange of information by diverse communities — bioinformatics, cultural heritage, ontologies, GIS and many others — has motivated the appearance of important applications for which the techniques developed until now are not effective, since there is neither a relatively regular structure that can be exploited, nor is the purely textual information predominant. We will term these collections *highly heterogeneous*. For example, in bioinformatics there is an enormous amount of information labeled in XML, ranging from molecular information to ontologies that describe the concepts of the domain, and the contents of technical articles. Many applications depend on the integrated use of those sources, but in that context the currently available techniques are not directly applicable. This causes enormous efforts in the creation of ad-hoc approaches

for each particular application.

The main reason for the inadequacy of current techniques is the lack of support for heterogeneity at the structural level: techniques developed for data-oriented collections support rich XML structures, but do not tolerate a high level of variation in them; conversely, techniques techniques developed for document-oriented collections are very good at processing complex textual content, but are limited in their support for complex structural information. This thesis is concerned with collections of XML documents with rich and complex structures, which are not well supported by either data- or document-oriented XML processing techniques because their heterogeneity is present at many different *levels of granularity*: text, elements, paths, subtrees and entire documents.

This scenario implies the need for *approximate* retrieval techniques that are tolerant of significant variability in document structures. In most cases there will be no exact answer to a given query, but a set of approximate results ranked according to a similarity function. A crucial issue is that a single notion of similarity that works "best" in any situation does not exist. Different users in different contexts may require different similarity functions; for instance, a biologist may wish to retrieve proteins based on a comparison with a given amino acid sequence, while another one may issue a query asking for "malaria antigen" in the associated textual description, and a third user may combine both kinds of queries. This leads to the notion of *multi-similarity* systems [Adalı et al., 1998], which are designed to support multiple notions of similarity simultaneously. These systems require the possibility to combine different similarity measures depending on the characteristics of the data that need to be handled.

The first step required to achieve this objective is the definition of a multi-similarity framework for defining flexible XML measures. We present a literature study that shows that many existing similarity-oriented approaches for XML depend on combinations of common functions that operate at different granularity levels. For instance, a given approach may consist of a Tree Edit Distance for computing structural similarity, which in turns relies on a $tf \times idf$-based term similarity function for matching element tags. We propose to integrate these measures into a common framework that allows the definition of new, complex measures by composition of predefined functions. An important advantage of this approach is that it simplifies the integration of external, domain-specific functions such as a protein similarity measure.

The second step is to define a query model for highly heterogeneous XML collections. Our approach is based on using XML subtree *patterns* as flexible models for the information needs of the users. These patterns are matched to the particular *fragments* in the collection that maximize the similarity according to a given measure. The method presented has two main advantages: first, it is able to match subtrees with very high structural diversity, in which even the parent/child relation between nodes (which most other approaches depend on) is not maintained; and second, it is independent of any particular XML similarity measure.

Our fragment-based similarity algorithm is presented in two flavors: an exhaustive

version which uses a tailored indexing structure to return all matches, and a top-k version. The latter is based on an *index-merge* framework [Xin et al., 2007], which is able to support non-monotonic measures; this is a significant advantage over existing top-k approaches for querying XML, which are based on techniques that can only handle monotonic measures. In contrast, our method is able to work with a much broader class of functions, which is an especially desirable property when combined with a flexible multi-similarity framework.

This thesis also develops auxiliary techniques that are important in order to be able to integrate the fragment-based similarity approach into practical XML management systems. At the query language level, we provide a specification for the integration of the techniques outlined above into standard XQuery, which has been implemented into an experimental prototype. At the query post-processing level, it is often the case when dealing with highly heterogeneous collections that result sets can be quite complex; a clustering algorithm adapted for these cases is presented.

Finally, all of these techniques have been implemented as part of *ArHeX*, a toolkit for the creation of multi-similarity systems. The main user-level elements of ArHeX are a graphical tool for the creation and management of component-based similarity measures, and a query tool that supports fragment-based queries, including the XQuery extensions described above. The implementation is supported by a measure metadata repository and Berkeley DB-based ondexing facilities.

1.3 Contributions

In summary, the main contributions of this thesis are the following:

- This thesis includes a review of the literature on approximate XML processing, which serves to motivate the need for new approaches for handling highly heterogeneous XML collections. In addition, we present a method to quantitatively characterize the heterogeneity of XML collections at several granularity levels.
- Multi-similarity systems require the ability to choose between a set of similarity measures, to tailor existing measures, and to create new ones on the fly. We present a new framework for similarity measures for XML that addresses these requirements: it is compositional in nature, allowing the integration of new similarity measures into existing systems and providing means to describe and parametrize the measures at any level. We propose a Description Logic-based formalism to describe these measures, which is significantly richer than existing approaches.
- We introduce *fragment-based queries,* a flexible query model that serves as a primitive for querying large-scale, highly heterogeneous XML databases. We show how fragment-based queries, as a query primitive, can deal with a much higher degree of flexibility than existing approaches.

- We present a top-k adaptation of the fragment-based query approach, which offers a significant performance improvement with respect to the exhaustive algorithm. A major advantage with respect to the existing ranking algorithms for XML is that it supports non-monotonic similarity measures; to the best of our knowledge, it is the first top-k XML approach that does so. Supporting a broad class of functions is particularly important given the flexibility provided by our framework for similarity measures.
- We produce a specification for the incorporation of similarity-based techniques into standard XQuery implementations based on a functional interface. This is a first step towards the deep integration of structural similarity-based approaches into XQuery systems.
- In our context, query result sets may be significantly heterogeneous themselves, which may be difficult to interpret. To mitigate this problem, we introduce a representative-based clustering algorithm that is useful for organizing such result sets.
- We present ArHeX, a toolkit that illustrates the application of the approaches presented in this thesis to the support of the engineering of multi-similarity systems.

1.4 Organization

This section describes the organization of the remainder of the thesis. A brief summary of each chapter is shown below.

Chapter 2. Background: Exact and Approximate Management of XML Data

This chapter briefly introduces the main XML-oriented data processing techniques. Then, it focuses on describing the state of the art in similarity-based XML applications by introducing the main approaches to compute similarity between XML documents. A classification that organizes these disparate approaches into a coherent framework is proposed. Finally, a precise characterization of "highly heterogeneous" XML collections using information-theoretic criteria is introduced.

Chapter 3. A multi-similarity framework for XML

A key observation presented in the previous chapter is that most of the XML similarity measures presented in the literature can be expressed as combinations of predefined, generic measures that operate at diverse granularity levels. For instance, $tf \times idf$ can be used for weighting textual items, and the tree edit distance can be used to compare structures. In this chapter we present a framework for the definition of generic XML

similarity measures, based on (*i*) an adaptation of the *composite pattern*, used in software engineering for creating complex object out of reusable components and (*ii*) metadata facilities based on Description Logic, which are useful for the organization of diverse measures in a multi-similarity system.

Chapter 4. A flexible fragment-based querying model for XML

This chapter presents *fragment-based queries,* a flexible primitive for approximate XML retrieval. The approach relies on the identification of the *fragments* of documents that are similar to a given *pattern*, which serves as an abstract representation of a user request, whose precise semantics is dependent of the particular similarity measure that must be applied for ranking. For example, depending on a particular measure, the particular hierarchy of elements, or the ordering of siblings, may or may not be deemed to be relevant when searching for occurrences in the database. This chapter introduces the fragment matching algorithms and its associated techniques, and presents an extensive experimental evaluation.

Chapter 5. Extensions

This chapter presents several refinements of fragment-based querying which are important for its practical usage. First of all, the chapter presents a specification detailing how fragment-based algorithms can be incorporated into an XQuery implementation. Then, it introduces a ranked top-k version of the fragment-based querying algorithm. In contrast with previously existing approaches, this algorithm does not require the similarity measure to be monotonic; this makes it usable with the generic measures introduced in chapter 3. Experimental results show a significant performance improvement with respect to the exhaustive algorithm presented in Chapter 4. The final contribution is an adaptation of a representative-based clustering algorithm adapted to XML collections; in particular, it is useful to organize the results of some flexible queries on highly heterogeneous collections, which can return apparently very disparate result sets.

Chapter 6. Prototype

The algorithms outlined in this thesis have been implemented as part of ArHeX, a toolkit for the engineering of multi-similarity XML systems. This chapter presents in detail a graphical query system and a component-based measure designer, in the context of a biomedical research project.

Chapter 7. Conclusions

The last chapter addresses conclusions and future work. A list of papers published as the result of this thesis work is included.

Chapter 2

Background: Exact and Approximate Management of XML Data

This chapter presents introductory material that provides context for the contributions of this thesis. Section 2.1 surveys the basic developments that underlie semi-structured and XML data management. Section 2.2 describes techniques commonly used in XML databases for indexing data. Section 2.3 presents approximate approaches used in XML databases, and in particular in surveys the similarity measures described in the literature. Finally, section 2.4 discuses the concept of *heterogeneity* in the context of XML databases, and proposes formal criteria to characterize the heterogeneity of XML collections.

2.1 Introduction to XML data management

This section introduces the basic concepts regarding the management of XML data. First, Section 2.1 provides a historical introduction to the main concepts in the field. Then, Section 2.1.2 focuses on XPath and XQuery, the standard languages for querying XML.

2.1.1 Semi-structured data management

XML data management systems are rooted in "web data"-oriented systems that appeared in the mid-90s [Abiteboul et al., 1999]. These systems are characterized by being tailored to "semi-structured" data [Abiteboul, 1997], defined by [Quass et al., 1997] as "data that has no absolute schema fixed in advance, and whose structure may be irregular or incomplete". The name was chosen to contrast both "structured" (i.e. relational and object-oriented) and "unstructured" (textual Information Retrieval) systems.

Semi-structured systems were built around a new data model representing data as labeled directed graphs, in which the nodes where labeled with meaningful descriptions of their content. The canonical semi-structured data model is OEM [Papakonstantinou et al., 1995], the *object exchange model*, in which objects were 4-tuples with the following fields:

$$< label, type, value, oid >$$

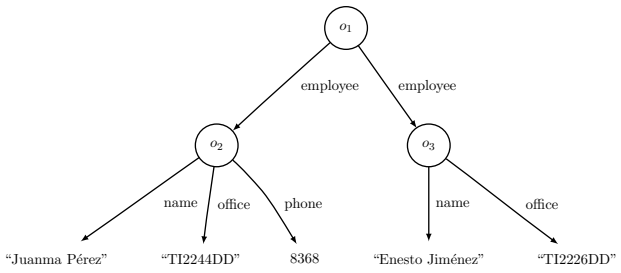


Figure 2.1: An example OEM database

where *label* is a string describing what this object represents, *type* is the data type of the value (lists and sets were allowed, in addition to atomic types), *value* is the value of the object and *oid* is an object identifier, or Λ (NULL).

The *oid* field was used to link objects together, forming directed graphs. For example, figure 2.1 shows a database with two objects; one of them (o_2) is:

$$< employee, set, \{o_4, o_5, o_6\}, o_2 >$$
$$< name, string, \text{“}Juanma\ Pérez'', o_4 >$$
$$< office, string, \text{“}TI2424DD'', o_5 >$$
$$< phone, int, 8368, o_6 >$$

Research into semistructured data produced experimental systems such as Lore [McHugh et al., 1997], which included a path-oriented query language (Lorel) on top of an algebraic optimization system. From the beginning it was recognize that *path expressions,* similar in principle to those in the object-oriented query language OQL, were an important primitive. A very simple Lorel query that incorporates two path expressions is:

```
select employee.office
where employee.name = ''Ernesto Jiménez''
```

In Lorel, OQL path expressions were augmented with new *wildcards*, in order to enable queries when the schema is not entirely known.

In order to speed up path traversals, a new type of index was devised: the *DataGuide*. Given an OEM database, a DataGuide provides a structural summary of the data stored in the database, computed using an algorithm that relies on the conversion between a non-deterministic finite automata into a deterministic one. It essentially generates a schema for the database, annotated with some extra information useful for optimization. Figure 2.2 shows a DataGuide which summarizes the OEM database in Figure 2.1. A slightly more complex example can be found in Figure 2.3.

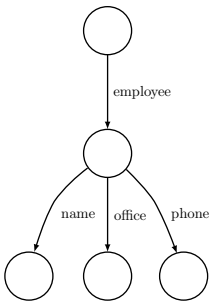


Figure 2.2: An example DataGuide

2.1.2 XML, XPath and XQuery

The road towards the standardization of XML in 1998 had progressed very much in parallel with these developments. Thus, "when XML came along, the similarity between OEM and XML was striking, and exciting" [Widom, 1999]. For example, the previous OEM object can be trivially transformed into XML:

```
<employee>
<name>Juanma Pérez</name>
<office>TI2424DD</office>
<phone>8368</phone>
</employee>
```

Nevertheless, some significant differences exist:

- XML nodes (*elements*) can have *attributes*, which are name-value pairs.
- Although in theory the XML data model is a directed graph like OEM, in practice this is seldom used; for all practical purposes, XML documents are *trees*.

A number of query languages for XML soon emerged, such as YATL [Christophides et al., 2000], XQL [Robie et al., 1988] and XML-QL [Deutsch et al., 1999]; Lorel itself was quickly adapted to work with XML. A comparison of these early languages can be found in [Fernández et al., 1999] and [Bonifati and Ceri, 2000].

In parallel, the World Wide Web Consortium released the specification of the XPath language, in order to address parts of XML documents – a requirement that had arisen during the development of several new XML-based specifications. Basically, XPath is based on path expressions with wildcards. Consider the following XPath expressions, that return the name of an employee in the XML version of our example database:

```
/employee/name
/*/name
```

Axis name	Action
ancestor	Selects all ancestors (parent, grandparent, etc.) of the context node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the context node and the context node itself
attribute	Selects all attributes of the context node
child	Selects all children of the context node
descendant	Selects all descendants (children, grandchildren, etc.) of the context node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the context node and the current node itself
following	Selects everything in the document after the closing tag of the context node
following-sibling	Selects all siblings after the context node
namespace	Selects all namespace nodes of the context node
parent	Selects the parent of the context node
preceding	Selects everything in the document that is before the start tag of the context node
preceding-sibling	Selects all siblings before the context node
self	Selects the context node

Table 2.1: XPath axes

Path expressions are lists of "location steps" that traverse a document along an *axis*; in the previous example, the *child* axis is traversed. The axes are defined with respect to a given node, named the *context node*. There are 14 axes defined in the standard, shown in Table 2.1.

It is also possible to restrict the returned nodes using predicates:

```
/employee[name=``Juanma Pérez'']/phone
```

Structural conditions can be specified within a predicate. This feature turns path expressions into subtree patterns, named *twigs*. Solving XPath queries with twigs requires a join operation between the results of two (or more) different path expressions.

XPath compatibility was one of the requirements for a standard XML query language. The final specification, XQuery [Boag et al., 2007], is based on a hybrid language, Quilt, described by [Chamberlin et al., 2000] in this way :

> Our strategy in designing the language has been to borrow features from several other languages that seem to have strengths in specific areas. From XPath and XQL we take a syntax for navigating in hierarchical documents. From XML-QL we take the notion of binding variables and then using the bound variables to create new structures. From SQL we take the idea of a series of clauses based on keywords that provide a pattern for restructuring data (the SELECT-FROM-WHERE pattern in SQL). From OQL we take

> the notion of a functional language composed of several different kinds of expressions that can be nested with full generality. We have also been influenced by reading about other XML query languages such as Lorel and YATL. We decided to name our language Quilt because of its heritage as a patchwork of features from other languages, and also because of its goal of assembling information from multiple diverse sources.

XQuery has become a successful recommendation, being incorporated into mainstream XML-capable DBMSs, including Oracle, DB2 and SQL Server.

2.2 Indexing and processing techniques for XML databases

XML indexes can be broadly classified in two families: *content* indexes, which speed up queries referring to the textual content of documents, and *structural* indexes which efficiently resolve location steps in path expressions. Few differences can be identified between the requirements of content indexes with respect to those found in Information Retrieval [Baeza-Yates and Ribeiro-Neto, 1999]; usually inverted lists are used, stored in a B^+-tree. In contrast, new XML-specific approaches have been developed for XML-specific structural indexes, which we will describe in this section.

The main requirement for a structural index for XML is to support all XPath axes. It should be remarked, however, that the most important axes by far are the ancestor/descendant axes. The key primitives that must be supported are:

- Find all descendants of a node,
- Find all ancestors of a node,
- Find all leaves of a document,
- Find the preceding nodes,
- Find the *nca* (nearest common ancestor) of two nodes.

Many indexing method that support these primitives have been proposed in the literature. They can be classified in two families: those approaches that are based on summarizing the structure of documents using some graph traversal technique, and those based on encoding nodes using a labeling scheme. The following sections present both paradigms.

2.2.1 Techniques based on graph traversal

A number of proposed indexes are based on *graph traversal*: they rely on a graph traversal of the database for storing a summary of its structures. The DataGuide is the classic example, and many refinements on its ideas have been developed. A particularly relevant

concept is *bisimulation*, that is an equivalence relation between state transition systems, associating systems which behave in the same way in the sense that one system simulates the other and vice-versa. There are many variations of this concept; for instance, [Kaushik et al., 2002] use the following definition:

Definition 1 (*bisimulation*) A symmetric, binary relation $\approx$ on a graph G is called a bisimulation if, for any two data nodes u and v with $u \approx v$, we have that (a) u and v have the same label, (b) if $parent(u)$ is the parent of u and $parent(v)$ is a parent of v, then $parent(u) \approx parent(v)$. Two nodes u and v in G are said to be bisimilar, denoted by $u \approx^b v$, if there is some bisimulation $\approx$ such that $u \approx v$. □

T-Index [Milo and Suciu, 1999] index a sequence of path expressions defined by a *template*. They define templates with the forms

$$T_1 x_1 T_2 x_2 \ldots T_n x_n$$

where T_i can be either a regular path expression or one of the placeholders $\boxed{P}$ and $\boxed{F}$. A query is obtained by replacing $\boxed{P}$ with a regular path expression, and $\boxed{F}$ with a formula (a predicate, in XPath nomenclature). The paper focuses on the special cases $t_1 = \boxed{P}x$ and ; the sequences produced are named 1- and 2-indexes respectively.

Instead of producing a deterministic finite automata like in the DataGuide approach, the T-index finds a bisimulation . Note, however, that for tree-shaped data the 1-index and the DataGuide are completely equivalent. Figure 2.3 shows an example extracted from [Milo and Suciu, 1999].

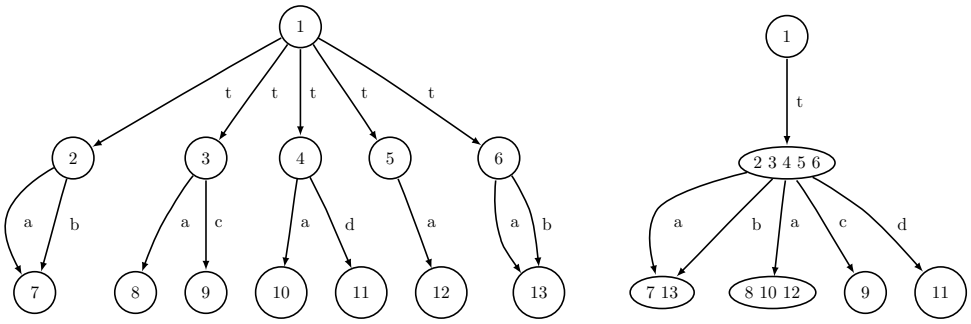
APEX [Chung et al., 2002] is an "adaptive path index for XML data". It performs a summarization of the most frequent paths in the query workload

Other indexing techniques, more oriented towards twig query processing, include the F&B index [Kaushik et al., 2002], ViST [Wang et al., 2003] and PRIX [Rao and Moon, 2004]. More complete surveys can be found in [Catania and Maddalena, 2002, Wong et al., 2006].

2.2.2 Techniques based on node labeling schemes

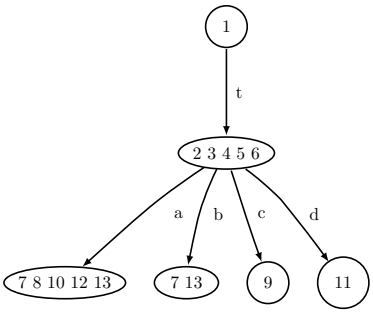
The alternative to graph traversal techniques are labeling schemes, which assign to each node in the XML tree one or more numeric identifiers, which allows the computation of relationships between nodes using simple arithmetic operations.

Following [Christophides et al., 2004], we will consider three families of labeling schemes: bit-vector, prefix and interval schemes. Figure 2.4 shows a simple example comparing several approaches from all three families.

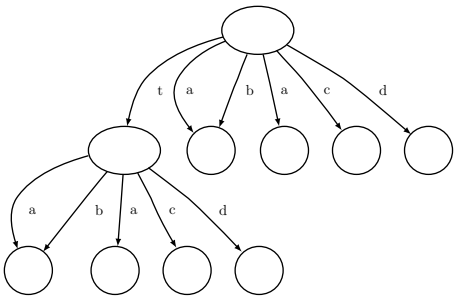


(a) Sample database

(b) 1-index



(c) DataGuide



(d) 2-index

Figure 2.3: T-Index compared to DataGuide

Bit-vector schemes In the bit-vector schemes representation, node labels are bit strings that encode structural information. The most basic case was proposed by [Wirth, 1988], whose algorithm encodes, in linear time, every node u using the function $l(u) = \{b_1, \ldots, b_k\}, b_i = 1$ if the i-th node is either u or an ancestor of u. Using this scheme, simple binary *and* and *or* operations suffice to check in constant time (given a pre-fixed k) if a node is an ancestor of another, and can also obtain the nearest common ancestor of two nodes. Many variations exist that produce more compact labels, typically having a complexity of $O(\log n)$ [Kaplan et al., 2002]. Unfortunately, the more compact variations are only capable of solving ancestor/descendant/sibling queries in $O(n)$.

Prefix schemes Prefix schemes encode the parent of an node as part of the label. The best know prefix scheme is Dewey's Decimal Coding System (DDC), commonly used in public libraries [Dewey, 2003]. In Dewey, the label of a node u is $l(v)l(u)$, where $l(v)$ is the label of its parent and $l(u) \in \{0, \ldots, 9\}$. Checking if a node is an ancestor of another one is reduced to a simple check for a common prefix, as well as obtaining the nearest common ancestor. Dewey's approach can be easily generalized to nodes with more than 10 children. As in the previous family, there are many variations, involving more compact labels (but at a higher cost).

Interval schemes In interval schemes, each node is assigned a $[start, end]$ interval that encodes useful structural information. Dietz [Dietz and Sleator, 1987] proposes using $l(v) = [pre(v), post(v)]$, where $pre(v)$ and $post(v)$ are the preorder and postorder numbers of v. Using this scheme, u is a descendant of v if $pre(v) < pre(u)$ and $post(v) > post(u)$. This scheme can be easily adapted to XML adding a third number that represents the *level*of the node in the document, which allows direct parent/child and leaf queries. For efficient sibling queries, the *parent* preorder number is added to the encoding instead of the level.

Many variations exist. [Agrawal et al., 1989] uses, instead of preorder and postorder numbers, $[index(v), post(v)]$, where $index(v)$ is the lowest postorder number of u's descendants. Meanwhile, [Li and Moon, 2001] uses $[pre(v), size(v)]$, where $size(v)$ is the size of the subtree rooted at v. All of these schemas are essentially equivalent for trees (but not when adapted for DAGs).

2.3 Approximate retrieval in XML databases

2.3.1 Survey of approximate techniques

An early approach for XML approximate retrieval is ELIXIR [Chinenyanga and Kushmerick, 2002] that allows approximate matching in data content (tree leaves), and ranks the results according to the matching degree, disregarding structure in the evaluation.

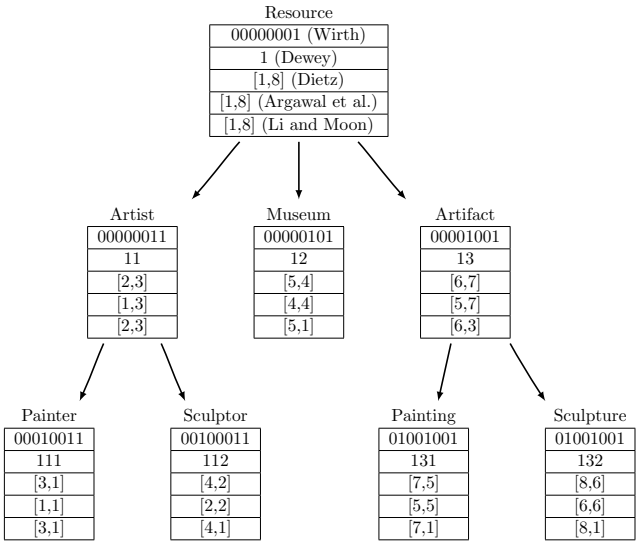


Figure 2.4: A comparison of labeling schemes from [Christophides et al., 2004].

No structural heterogeneity is considered, and vocabulary heterogeneity is allowed only for data content elements, not for tags.

More sophisticated approaches, like XIRQL [Fuhr and Großjohann, 2001] and XXL [Theobald and Weikum, 2002], accept approximate matching at nodes and then discuss how to combine weights depending on the structure. Vocabulary heterogeneity is supported for content and element tags, but no structural heterogeneity is allowed: conditions on document structure are interpreted as filters, thus they need to be exactly satisfied. XXL supports a similarity operator and, to use this operator, the user should be aware of the occurrence of similar keywords or element tags.

In the approach proposed by Damiani and Tanca [Damiani and Tanca, 2000] both XML documents and queries are modeled as graphs labeled with fuzzy weights capturing the information relative relevance. They propose to employ both structure related weighting (weight on an edge) and tag related weighting (weight on a node). Some criteria for weighting are proposed such as the weight decreases as moving far away from the root, the weight depends on the dimension of the subtree. Shortcut edges are considered, thus allowing the insertion of nodes, which weight is function of weights of edges. The match score is a normalized sum of weight of edges.

In the tree relaxation approach [Amer-Yahia et al., 2002] exact and relaxed weights are associated with query nodes and edges. The score of a match is computed as the sum of the corresponding weights, and the relaxed weight is function of the transformations applied to the tree. The considered transformations are: *relax node*, replacing the node content with a more general concept; *delete node*, making a leaf node optional by removing

	Vocabulary heterogeneity	Structural heterogeneity	Ranking
ELIXIR	on content	–	structure independent
XIRQL	on content	–	structure dependent
XXL	on content and tag	–	structure dependent
[Damiani and Tanca, 2000]	on content and tag	node insertion	structure dependent
[Amer-Yahia et al., 2002]	on content and tag	delete node, relax edge, promote node	structure dependent
approXQL	on content and tag	node insertion, node deletion	structure dependent
[Amer-Yahia et al., 2005]	on content and tag	edge generalization, leaf deletion, subtree promotion	structure dependent

Table 2.2: Heterogeneity degree in XML approximate querying

the node and the edge linking it to its parent; *relax edge*, transforming a parent/child relationship to an ancestor/descendant relationship; *promote node*, moving up in the tree structure a node (and the corresponding subtree).

The approXQL [Schlieder, 2001] approach can also handle partial structural matchings. All the paths in the query are however required to occur in the document. The allowed edit operations on the document tree are delete node, insert intermediate node, relabel node. The score of match is function of the number of transformations, each one of which is assigned with a user-specified cost.

Amer-Yahia et al. in [Amer-Yahia et al., 2005] account for both vocabulary and structural heterogeneity and propose scoring methods that are inspired by $tf * idf$ and rank query answers on both structure and content. Specifically, twig scoring accounts for all structural and content conditions in the query. Path scoring, by contrast, is an approximation of twig scoring that loosens the correlations between query nodes when computing scores. The key idea in path scoring is to decompose the twig query into paths, independently compute the score of each path, and then combine these scores.

The main differences among the considered approaches are summarized in Table 2.2. Note how approaches to XML approximate queries have progressively shifted to higher degrees of heterogeneity, and to cope also with structural heterogeneity. We remark that all the considered approaches enforce at least the ancestor-descendant relationship in pattern retrieval.

Tree Embedding and Top-k Processing A different perspective from which approaches to XML approximate querying started, is that of looking for approximate structural matches, allowing the structure of the document and query trees to partially match. [Kilpeläinen, 1992] discusses ten different variations of the tree inclusion problem, that is, the problem of locating instances of a query tree Q in a target tree T. He orders these different problems, ranging from unordered tree inclusion to ordered subtrees, highlighting the inclusion relationships among them. Moreover, he gives a general schema of solution,

and instantiates it for each problem, discussing the resulting complexities. His goal, however, is not to measure the distance, nor to rank results. All the instances of the pattern in the data tree are returned.

Another approach aimed at identifying the matches but that does not rank them, is by [Kanza and Sagiv, 2001]. They advocate the need of more flexible query evaluation mechanisms in the context of semi-structured data, where both queries and data are modeled as graphs. They propose mapping query paths to data paths, so long as the data path includes all the labels of the query path; the inclusion needs not to be contiguous or in the same order.

Sub-optimal approaches for ranked tree-matching have been proposed for dealing with the NP complexity of the tree inclusion problems widely discussed and analyzed in [Kilpeläinen, 1992]. In these approaches, instead of generating all (exponentially many) candidate sub-trees, the algorithms return a ranked list of "good enough" matches. For example, in [Schlieder and Naumann, 2000] a dynamic programming algorithm for ranking query results according to a cost function is proposed. In [Amer-Yahia et al., 2002] a data pruning algorithm is presented where intermediate query results are filtered dynamically during evaluation process. ATreeGrep [Shasha et al., 2002], instead, uses as basis an exact matching algorithm, but allowing a fixed number of "differences" in the result. [Marian et al., 2005] proposes an adaptive query processing algorithm to evaluate approximate matches where approximation is defined by relaxing XPath axes. In adaptive query processing, different plans are permitted for different partial matches, taking the top-k nature of the problem into account.

2.3.2 Similarity measures for XML data

In addition to the general approaches just described, a number of XML-specific similarity measures have been proposed. In most cases, these measures reflect the fact that XML documents are hierarchical in nature and can be viewed as compositions of simpler constituents, including elements, attributes, links, and plain text. The hierarchy of composition is quite rich: attributes and texts are contained in elements, and elements themselves are organized in higher-order structures such as paths and subtrees. We will refer to each level in the compositional structure of an XML document as a *granularity level*. Section 2.3.2.1 introduces the levels which are used in the literature; they are useful in analyzing the diverse XML similarity measures that have been proposed in the literature.

The rest of the section surveys a number of XML similarity measures. Most fall into two categories: *tree-based* and *vector-based* approaches. Tree-based approaches exploit the tree structure of XML, applying graph-theoretic techniques to devise similarity functions. In contrast, *vector-based* approaches adapt models commonly used in Information Retrieval, consisting in defining measures over vector-based representations of XML documents.

Sections 2.3.2.2 and 2.3.2.3 surveys tree-based and vector-based measures respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<recipes>
<summary> Some recipes of my GranMa </summary>
<recipe num="1">
<title>Pizza Margherita</title>
<preparation>
<ingredients>
<ingredient name="Tomato"
            amount="1" unit="Kg"/>
<ingredient name="Mozzarella"
            amount="3" unit="pieces"/>
...
</ingredients>
<step> Preheat oven to 180 degrees C</step>
...
<step>...</step>
</preparation>
<cost amount="5" unit="euro"/>
<note>Chunk Tomato in
little pieces, ...</note>
</recipe>
...
</recipes>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<collection>
<description> Some recipes of Aunt Carol
</description>
<recipe>
<name>Pizza Margherita</name>
<ingredient name="Tomato" qty="1.5"/>
<ingredient name="Mozzarella" qty="2"/>
...
<preparation>
<step> Preheat oven to 180 degrees C</step>
<step>...</step>
...
</preparation>
<comment>
Chunk Tomato in little pieces, ...
</comment>
<nutrition calories="500"
           fat="20" protein="18"/>
</recipe>
...
</collection>
```

Figure 2.5: Sample XML documents containing recipes

Section 2.3.2.4 describes other approaches that do not fit neatly into the previous categories. All of the the examples in these sections are based on the documents shown in Figure 2.5.

2.3.2.1 Granularity

[Guerrini et al., 2006] presents a study of the XML granularity levels that occur in the literature regarding similarity-oriented XML techniques. These are:

- the whole XML document,
- subtrees (i.e., portions of documents),
- paths,
- elements,
- links,
- attributes,
- textual content (of attributes and data content elements).

The relationships between the granularity levels are depicted in Figure 2.6 through arrows. An arrow from a granularity level A to a granularity level B means that a similarity measure at level A can be formulated in terms of objects at granularity B. Similarity measures for XML are usually defined according to these natural relations of composition. For instance, a measure for complete XML documents can be defined by

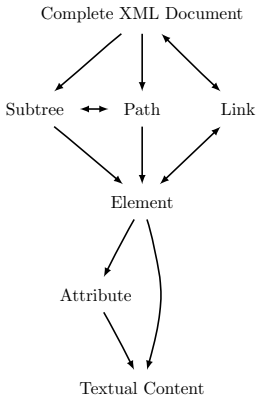


Figure 2.6: Levels of granularity in XML documents

evaluating the similarity of paths, which in turn requires some criterion to compare the elements contained in the path. In addition to composition, other relationships among elements/documents that can be exploited for measuring structural similarity include:

- parent-children relationship, that is the relationship between each element and its direct subelements/attributes;
- ancestor-descendant relationship, that is the relationship between each element and its direct and indirect subelements/attributes;
- order relationship among siblings;
- link relationship among documents/elements.

2.3.2.2 Tree-based approaches

In this section we deal with approaches for measuring the similarity between XML documents that rely on a tree representations of the documents. We first discuss the document representation as trees, the basics of measures for evaluating tree similarity, and then approaches specifically tailored to XML.

Tree Similarity Measures The problem of computing the distance between two trees, also known as tree editing problem, is the generalization of the problem of computing the distance between two strings [Wagner and M.J.Fisher, 1974] to labeled trees. The editing operations available in the tree editing problem are changing (i.e., relabeling), deleting, and inserting a node. To each of these operations a cost is assigned, that can depend on the labels of the involved nodes. The problem is to find a sequence of such operations

transforming a tree T_1 into a tree T_2 with minimum cost. The distance between T_1 and T_2 is then defined to be the cost of such a sequence.

The best known and reference approach to compute edit distance for ordered trees is [Zhang et al., 1989]. They consider three kinds of operations for ordered labeled trees. *Relabeling* a node n means changing the label on n. *Deleting* a node n means making the children of n become the children of the parent of n and then removing n. *Inserting* n as the child of m will make n the parent of a consecutive subsequence of the current children of m. Let Σ be the node label set and let λ be a unique symbol not in Σ, denoting the null symbol. An edit operation is represented as $a \rightarrow b$, where a is either λ or the label of a node in T_1 and b is either λ or the label of a node in T_2. An operation of the form $\lambda \rightarrow b$ is an insertion, an operation of the form $a \rightarrow \lambda$ is a deletion. Finally, an operation of the form $a \rightarrow b$, with $a, b \neq \lambda$ is a relabeling. Each edit operation $a \rightarrow b$ is assigned a cost, that is, a nonnegative real number $\gamma(a \rightarrow b)$ by a cost function γ. Function γ is a distance metric, that is:

$$\begin{aligned} \gamma(a \rightarrow b) &\geq 0 \\ \gamma(a \rightarrow a) &= 0 \\ \gamma(a \rightarrow b) &= \gamma(b \rightarrow a) \\ \gamma(a \rightarrow c) &\leq \gamma(a \rightarrow b) + \gamma(b \rightarrow c) \end{aligned}$$

Function γ is extended to a sequence of edit operation $S = s_1, \ldots, s_k$ s.t.. $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$.

The edit distance between the two trees T_1 and T_2 is defined as the minimum cost edit operation sequence that transforms T_1 to T_2, that is:

$D(T_1, T_2) = min_S\{\gamma(S) | S$ is an edit operations sequence taking T_1 to $T_2\}$.

The edit operations give rise to a mapping which is a graphical specification of which edit operations apply to each node in the two trees. Figure 2.7 is an example of mapping showing a way to transform T_1 to T_2. It corresponds to the edit sequence $name \rightarrow \lambda; calories \rightarrow at; \lambda \rightarrow preparation$.The figure also shows a left-to-right postorder of nodes which is commonly used to identify nodes in a tree.

Definition 2 (*Tree mapping*) For a tree T, let $t[i]$ represent the i-the node of T. A mapping (or matching) from T_1 to T_2 is a triple $(M, T_1, T_2)$where M is a set of pairs of integers (i, j) such that:

$1 \leq i \leq |T1|, 1 \leq j \leq |T_2|$;

for any pair $(i_1, j_1)$and (i_2, j_2)in M:

- $i_1 = i_{2:} \iff j_1 = j_2$(one-to-one),
- $t_1[i_1]$is to the left of $t_1[i_2]$ iff $t_2[j1]$is to the left of $t_2[j_2]$ (sibling order preserved),

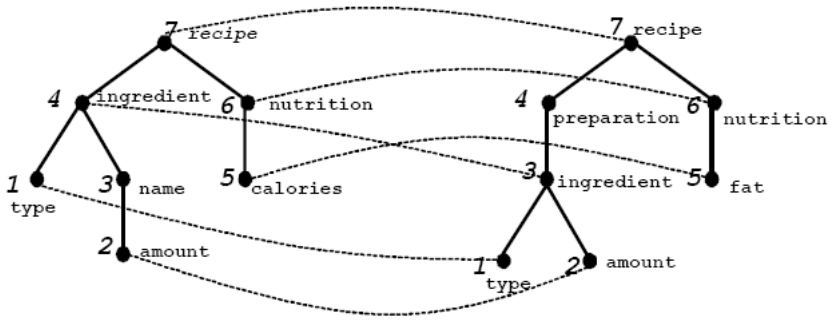


Figure 2.7: Mapping between two trees

- $t_1[i_1]$is an ancestor of $t_1[i_2]$ iff $t_2[j_1]$ is an ancestor of $t_2[j_2]$ (ancestor order preserved).

□

The mapping graphically depicted in Figure 2.7 consists of the pairs $(7,7)$, $(4,3)$, $(1,1)$, $(2,2)$, $(6,6)$, and $(5,5)$. Let M be a mapping from T_1 to T_2, the cost of M is defined as:

$$\gamma(M) = \sum_{(i,j)\in M} \gamma(t_1[i] \rightarrow t_2[j]) + \sum_{\{i|\neg\exists js.t.(i,j)\in M\}} \gamma(t_1[i] \rightarrow \lambda) + \sum_{\{i|\neg\exists js.t.(i,j)\in M\}} \gamma(\lambda \rightarrow t_2[j])$$

There is a straightforward relationship between a mapping and a sequence of edit operations. Specifically, nodes in T_1 not appearing in M correspond to deletions; nodes in T_2 not appearing in Mcorrespond to insertions; nodes that participate to M correspond to relabellings if the two labels are different, to null edits otherwise.

Different approaches [Selkow, 1977, Chawathe et al., 1996, Chawathe, 1999] to determine tree edit distance have been proposed as well. They rely on similar tree edit operations with minor variations. Table 2.3 [Dalamagas et al., 2006] summarizes the main differences among the approaches. The corresponding algorithms are all based on similar dynamic programming techniques. The [Chawathe, 1999] algorithm is based on the same edit operations (i.e., insertion and deletion at leaf nodes and relabeling at any nodes) considered by [Selkow, 1977] but it significantly improves the complexity by reducing the number of recurrences needed, through the use of edit graphs.

XML Specific Approaches The basic ideas discussed above for measuring the distance among two trees have been specialized to the XML context by the following approaches.

[Nierman and Jagadish, 2002] introduce an approach to measure the structural similarity specifically tailored for XML documents with the aim of clustering together documents presumably generated from the same DTD. Since the focus is strictly on structural

	Edit operations	Complexity
[Selkow, 1977]	insert node*, delete node*, relabel node	$4^{min(N \cdot M)}M$,N numbers of nodes of the trees
[Zhang et al., 1989]	insert node, delete node, relabel node	$O(M \cdot N \cdot b \cdot d)M, N$ numbers of nodes of the trees, b, d depths of the trees
[Chawathe et al., 1996]	insert node*, delete node*, relabel node, move subtree	$O(N \cdot D)N$ numbers of nodes of both trees, D number of misaligned nodes
[Chawathe, 1999]	insert node*, delete node*, relabel node	$O(M \cdot N)M, N$ dimension of the matrix that represents the edit graph

Table 2.3: Tree Edit Distance algorithms. Operations marked with an asterisk are restricted to leaves.

similarity, the actual values of document elements and attributes are not represented in their tree representations (i.e., leaf nodes of the general representation are omitted from the tree). They suggest to measure the distance between two ordered labeled trees relying on a notion of tree edit distance. However, two XML documents produced from the same DTD may have very different sizes due to optional and repeatable elements. Any edit distance that permits changes to only one node at a time will necessarily find a large distance between such a pair of documents, and consequently will not recognize that these documents should be clustered together as being derived by the same DTD.

Thus, they develop an edit distance metric that is more indicative of this notion of structural similarity. Specifically, in addition to insert, delete, and relabel operations of [Zhang et al., 1989], they also introduce the insert subtree and delete subtree editing operations, allowing the cutting and pasting of whole sections of a document. Specifically, operation $insertTreeT(A, i)$ adds A as a child of T at position $i + 1$ and operation $deleteTreeT(Ti)$ deletes T_i as the i-th child of T. They impose however the restriction that the use of the $insertTree$ and $deleteTree$ operations is limited to when the subtree that is being inserted (or deleted) is shared between the source and the destination tree. Without this restriction, one could delete the entire source tree in one step and insert the entire destination tree in a second step, thus making completely useless insert and delete operations. The subtree A being inserted/deleted is thus required to be contained in the source/destination tree T, that is, all its nodes must occur in T, with the same parent/child relationships and the same sibling order; additional siblings may occur in T (to handle the presence of optional elements), as graphically shown in Figure 2.8. A

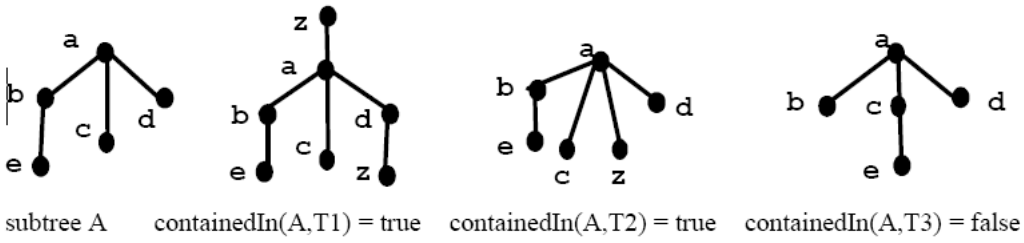


Figure 2.8: ContainedIn relationship

second restriction imposes that a tree that has been inserted via the $insertTree$ operation cannot subsequently have additional nodes inserted, and, analogously, a tree that have been deleted via the $deleteTree$ operation cannot previously had had nodes deleted. This restriction provides an efficient means for computing the costs of inserting and deleting the subtrees found in the destination and source trees, respectively. The resulting algorithm is a simple bottom up algorithm obtained as an extension of Zhang and Shasha's basic algorithm, with the difference that any subtree T_i has a graft cost which is the minimum among the cost of a single $insertTree$ (if allowable) and of any sequence of insert and (allowable) $insertTree$ operations, and similarly any subtree has a prune cost.

[Lian et al., 2004] propose a similarity measure for XML documents which, though based on a tree representation of documents, is not based on the tree edit distance. Given a document D they introduce the concept of *structure graph* (or s-graph) of D, $sg(D) = (N, E)$, as a direct graph such that N is the set of all elements and attributes in the documents of D and $(a, b) \in E$ if and only if a is in the parent-child relationship with b. The notion of structure graph is very similar to that of DataGuide presented in Section 2.1. Figure 2.9(b) shows the s-graph of the document in Figure 2.9(a). The similarity between two documents D_1 and $D_2$is then defined as

$$Sim(D_1,D_2) = \frac{|sg(D_1) \cap sg(D_2)|}{max\{|sg(D_1)|, |sg(D_2)|\}}$$

where $|sg(D_1)|$ is the cardinality of edges in $sg(D_1)$ and $|sg(D_1) \cap sg(D_2)|$ is the set of common edges between $sg(D_1)$ and $sg(D_2)$. Relying on this metric, if the number of common parent-child relationships between D_1 and D_2 is large, the similarity between the s-graphs will be high, and vice-versa. Since the definition of s-graph can be easily applied to set of documents, the comparison of a document with respect to a cluster can be easily accomplished by means of their corresponding s-graphs. However, as outlined by [Costa et al., 2004], a main problem with this approach relies on the loose-grained similarity which occurs. Indeed, two documents can share the same s-graph, and still have significant structural differences. Thus, the approach fails in dealing with application domains, such as wrapper generation, requiring finer structural dissimilarities. Moreover,

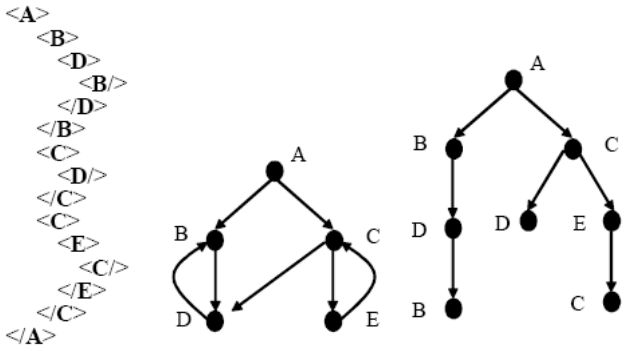


Figure 2.9: (a) The structure of a document, (b) its s-graph, (c) its structural summary

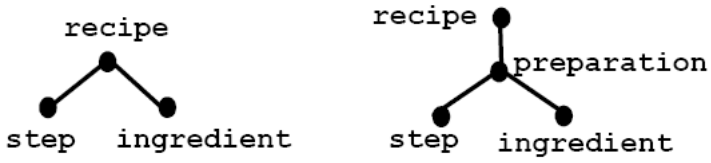


Figure 2.10: Two simple *s*-graphs

the similarity between the two *s*-graphs in Figure 2.10 is zero according to their definition. Thus the measure lacks to consider similar documents that do not share common edges even if they have some elements with the same labels.

[Dalamagas et al., 2006] presents an approach for measuring the similarity between XML documents modeled as rooted ordered labeled trees. The motivating idea is the same of [Nierman and Jagadish, 2002], that is, that XML documents tend to have many repeated elements and thus they can be large and deeply nested and even if generated from the same DTD can have quite different size and structure. Starting from this idea, the approach of [Dalamagas et al., 2006] is based on extracting structural summaries from documents by nesting and repetition reductions. Nesting reduction consists in eliminating non-leaf nodes whose label is the same of the one of its ancestor. By contrast, repetition reduction consists in eliminating, in a pre-order tree traversal, nodes whose path (starting from the root down to the node itself) has already been traversed. Figure 2.9(c) shows the structural summary of the document structure in Figure 2.9(a). The similarity between two XML documents is then the tree edit distance computed through an extension of the basic [Chawathe, 1999]. They claim, indeed, that using insertions and deletions only at leaves fits better in the XML context.

2.3.2.3 Vector-based approaches

In this section we deal with approaches for measuring the similarity that rely on a vector representations of documents. We first discuss the possible document representations as vectors, the different measures that can be exploited for evaluating vector similarity and then present some approaches specifically tailored to XML.

Document Representation Vector-based techniques represent objects as vectors in an abstract n-dimensional feature space. Let $O = (o_1, \ldots, o_m)$ be a collection of m objects; in our context, these can be whole XML documents, but also paths, individual elements, text, or any other component of a document. Each object is described in terms of a set of features $F = (F_1, \ldots, F_n)$, where each feature $F_i, i \in [1, n]$, has an associated domain Di which defines its permissible values. For instance, the level of an element is a feature whose domain is the positive integers (0 for the root, 1 for first-level elements, and so on). Feature domains can be either quantitative (continuous or discrete) or qualitative (nominal or ordinal). An object $o \in O$ is described as a tuple $(F_1(o), \ldots, F_n(o))$, where each $F_i(o) \in D_i$.

Consider for instance the two documents in Figure 2.5; we can represent them taking the elements as the objects to be the compared. The simplest possible feature is just the label of the document element, whose domain is a string according to the standard XML rules; in this case the roots of both documents are just described as the tuples (recipes) and (collections) respectively. Of course, other features are usually considered, possibly of different structural granularities. A typical example is the path to the root; for example, consider the leftmost ingredient element in each document. Both can be represented using the label and the path as features:

$F_{ingredient1} = (`ingredient', `/recipes/recipe/preparation/ingredients')$

$F_{ingredient2} = (`ingredient', `/collection/recipe')$

Some authors suggest restricting the length of the paths to avoid a combinatorial explosion. For example, [Theobald and Weikum, 2002] use paths of length 2.

Another important feature of elements is the k-neighborhood, that is, the set of elements within distance k of the element. For example, consider the 1-neighborhood (that is, parent and children) of the ingredient elements:

$F_{ingredient1} = (`ingredient', `ingredients', `name', `amount', `unit')$

$F_{ingredient2} = (`ingredient', `recipe, `name', `qty')$

Many variations are possible; for example, one of the components of the Cupid system by [Madhavan et al., 2001] uses as features the label, the vicinity (parent and immediate siblings), and the textual contents of leaf elements.

Vector-based Similarity Measures Once the features have been selected, the next step is to define functions to compare them. Given a domain D_i a comparison criterion for values in D_i is defined as a function $C_i : D_i \times D_i \rightarrow G_i$, where $G_i$is a totally ordered set,

typically the real numbers. The following property must hold: $C_i(f_i, f_i) = max_{y \in G}\{y\}$, that is, when comparing a value with itself the comparison function yields the maximum possible result. The simplest example of a comparison criterion is strict equality:

$$C_i(f_i, f_j) = \begin{cases} 1 & \text{if } f_i = f_j \\ 0 & \text{otherwise} \end{cases}$$

A similarity function as $S : (D_1, \ldots, D_n) \times (D_1, \ldots, D_n) \rightarrow L$, where L is a totally ordered set, can now be defined, that compares two objects represented as feature vectors and returns a value that corresponds to their similarity. An example of a similarity function is the weighted sum, which associates a weight w_i (such that $w_i \in [0, 1]$, $\sum_{i=1}^{n} w_i = 1$) to each feature:

$$S(o, o') = \frac{1}{n} \sum w_i C_i(f_i(o), f_i(o'))$$

If feature vectors are real vectors, metric distances induced by norms are typically used. The best-known examples are the L_1(Manhattan) and L_2(Euclidean) distances. Other measures have been proposed based on the geometric and probabilistic models.

The most popular geometric approach to distance is the vector space model used in Information Retrieval [Salton, 1983]. Originally it was intended to be used to compare the similarity among the textual content of two documents, but for the XML case it has been adapted for structural features as well.

The similarity in vector space models is determined by using associative coefficients based on the inner product of the document vectors, where feature overlap indicates similarity. The inner product is usually normalized, since, in practice, not all features are equally relevant when assessing similarity. Intuitively, a feature is more relevant to a document if it appears more frequently in it than in the rest of documents. This is captured by $tf \times idf$ weighting. Let $tf_{i,j}$ be the number of occurrences of feature i in document j, $df_i$the number of documents containing i, and N the total number of documents. The $tf \times idf$ weight of feature i in document j is:

$$w_{i,j} = tf_{i,j} \log \frac{N}{df_i}$$

The most popular similarity measure is the cosine coefficient, which corresponds to the angle between the vectors. Other measures are the Dice and Jaccard coefficients

$$\cos(u, v) = \frac{\mathbf{uv}}{|\mathbf{u}||\mathbf{v}|}$$

$$Dice(u, v) = \frac{2\mathbf{uv}}{|\mathbf{u}|^2|\mathbf{v}|^2}$$

$$Jac(u,v) = \frac{\mathbf{uv}}{|\mathbf{u}|^2|\mathbf{v}|^2 - \mathbf{uv}}$$

Another vector-based approach considers the objects as probability mass distributions. This requires some appropriate restrictions on the values of the feature vectors $(f_1, \ldots f_n)$; namely, all values must be nonnegative reals, and $\sum_{i=1}^{n} f_i = 1$. Intuitively, the value of f_i is the probability that the feature F_i is assigned to the object. In principle, correlation statistics can be used to compare distributions. The most popular are Pearson's and Spearman's correlation coefficients and Kendall's τ [Sheskin, 2003]. In addition, some information-theoretic distances have been widely applied in the probabilistic framework, especially the relative entropy, also called the Kullback-Leibler divergence.

$$KL(p_k||q_k) = \sum p_k \log_2 \frac{p_k}{q_k}$$

where p_k and q_k are the probability functions of two discrete distributions. Another measure of similarity is the mutual information.

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

where $P(x,y)$ is the joint probability density function of x and y (i.e., $P(x,y) = Pr[X = x, Y = y]$), and $P(x)$ and $P(y)$are the probability density functions of x and y alone.

An important use of information-theoretical measures is to restrict the features and objects to be included in similarity computations, by considering only the most informative. For example, [Theobald and Weikum, 2002] use the Kullback-Leibler divergence to cut down the number of elements to be compared in an XML classification system.

XML Specific Approaches Standard vector based approaches previously presented can easily be applied to XML documents whenever clustering is performed on a single granularity (e.g., clustering based on contents, on elements, or on paths). Specifically tailored approaches have been developed for XML documents that take more than one granularity along with their relationships into account. In these cases, given C the number of granularities, documents are represented through a C-dimensional matrix M in an Euclidean space based on one of two models, Boolean and weighted. With the Boolean model, $M(g_1, \ldots, g_C) = 1$ if the feature corresponding to the matrix intersection among granularities $g_1, \ldots, g_C$ exists, $M(g_1, \ldots, g_C) = 0$ otherwise. With the weighted model, $M(g_1, \ldots, g_C)$ is the frequency of the feature corresponding to the matrix intersection among granularities. Figure 2.11 displays a 3-dimensional Boolean matrix on granularities (document, path, term) stating the presence (or absence) of a term w_j in the element reached by a path P_j in a document D_m. As suggested by [Liu et al., 2004], once the

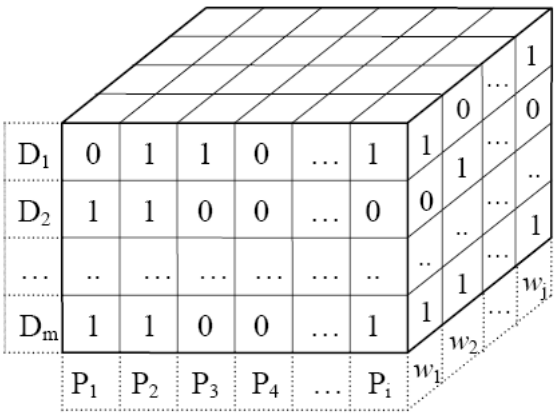


Figure 2.11: A 3-dimensional Boolean matrix

documents have been represented in the Euclidean space, standard approaches can be applied for measuring their similarity and create clusters. The big issue that should be faced is that the matrix can be sparse. Therefore, approaches for reducing the matrix dimension should be investigated along with the possibility to obtain approximate results.

According to our classification, [Yoon et al., 2001] propose a Boolean model with granularities (document, path, term) in which the path is a root-to-leaf path. A document is defined as a set of (p, v) pairs, where p denotes a root-to leaf path (named *ePath*) and v denotes a word or a content for an ePath. A collection of XML documents is represented through a 3-dimensional matrix, named *BitCube,* $BC(d, p, v)$, where D denotes a document, p denotes an ePath, v denotes word or content for p, and $BC(D, p, v) = 1$ or 0 depending on the presence or absence of v in the ePath p in D. The distance between two documents is defined through the Hamming Distance as

$$Sim(D_1, D_2) = |xor(BC(D_1), BC(D_2))|$$

where, *xor* is a bit-wise exclusive *or* operator applied on the representations of the two documents in the BitCube.

According to our classification, [Yang et al., 2005] exploit a weighted model with granularities (document, element, term). They employ the Structured Link Vector Model (SLVM) to represent XML documents. In the model of SLVM, each document, D_x in a document collection C, is represented as a matrix $d_x \in R^{n \times m}$, such that, $d_x =< d_{x(1)}, \ldots, d_{x(n)} >^T$ and $d_x(i) =< d_{x(i,1)}, \ldots, d_{x(i,m)} >$, where m is the number of elements, $d_{x(i,1)} \in R^m$ is a feature vector related to the term w_i for all subelements, $d_{x(i,j)}$is a feature related to the term w_i and specific to the element e_j, given as $d_{x(i,j)} = TF(w_i, doc_x.e_j) \cdot IDF(w_i)$ and $TF(w_i, doc_x.e_j)$ is the frequency of the term w_i in the element e_j of the document D_x, $IDF(w_i)$ is the inverse document frequency of w_i based

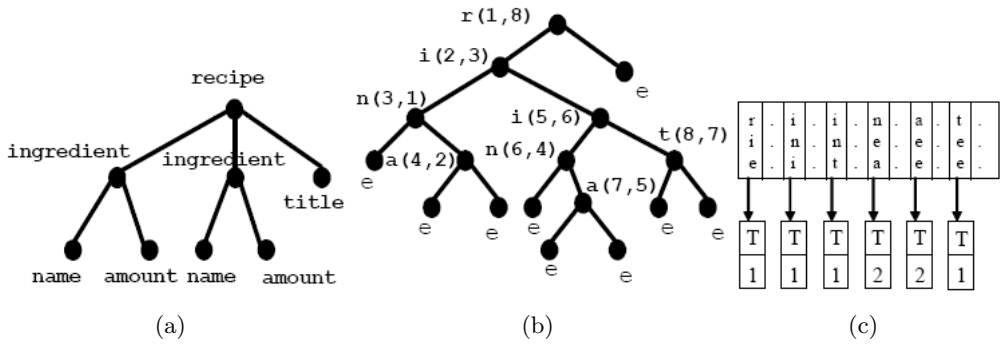


Figure 2.12: (a)A tree document, (b)its full binary tree, and (c) the binary branch vector

on C (each $d_{x(i,j)}$ is then normalized by $\sum_i d_{x(i,j)}$). The similarity measure between two documents D_x and D_y is then simply defined as

$$Sim(D_x, D_y) = \cos(d_x, d_y) = d_x \cdot d_y = \sum_{i=}^{n} d_{x(i)} d_{y(i)}$$

Where ·indicates the vector dot product, and d_x, d_y are the normalized document feature vectors of D_x and D_y. A more sophisticated similarity measure is also presented by introducing a kernel matrix

$$Sim(D_x, D_y) = \sum_{i=1}^{n} d_x^T \centerdot M_e \centerdot d_{y(i)}$$

where M_e is a $m \times m$ kernel matrix which captures both the similarity between pairs of elements as well as the contribution of a pair to the overall similarity. An entry in M_e being small means that the two elements should be semantically unrelated and some words appearing in the two elements should not contribute to the overall similarity and vice versa. An interactive estimation procedure has been proposed for learning a kernel matrix which captures both the element similarity and the element relative importance.

[Yang et al., 2005] propose an approach for determining a degree of similarity between a pair of documents that it is easier to compute with respect to tree edit distance and forms a lower bound for the tree edit distance. Their approach thus allows to filter out very dissimilar documents and computes the tree edit distance only with a restricted number of documents. Starting from a tree representation of XML documents (as the one in Figure 2.12(a)), they represent them as standard full binary trees (Figure 2.12(b)). A full binary tree is a binary tree in which each node has exactly zero or two children (the first child represents the parent-child relationship, whereas the second child represents the sibling relationship).Whenever one of the children is missing, it is substituted with ε.

The binary branch of the full binary tree (i.e. all nodes with their direct children) are then represented in a binary branch vector, $BRV(D) = (b_1, \ldots, b_\Gamma)$, in which b_i represents the number of occurrences of the i-th binary branch in the tree, Γ is the size of the binary branch space of the data set. The binary branch vector for the document in Figure 9(a) is shown in Figure 2.12(c). The binary branch distance between XML documents D_1 and D_2, such that $BRV(D_1) = (b_1, \ldots, b_\Gamma)$, and $BRV(D_2) = (b'_1, \ldots, b'_\Gamma)$, is computed though the Manhattan distance:

$$BDist(D_1, D_2) = ||BRV(D_1) - BRV(D_2)||_1 = \sum_{i=1}^{\Gamma} |b_i - b'_i|$$

In this approach the authors consider three granularities (element, element, element) that are bound by the parent-child and the sibling relationships. Then thanks to the transformation of the document tree structure in a full binary tree structure, they are able to use a 1-dimensional vector for the representation of a document.

2.3.2.4 Other approaches

We now present some approaches for evaluating similarity that do not fit in the vector-based or tree-based representation of documents.

Time series based approach. [Flesca et al., 2002] represent the structure of an XML document as a time series in which each occurrence of a tag corresponds to a given impulse. Thus, they take into account the order in which tags appear in the documents. They interpret an XML document as a discrete-time signal in which numeric values summarize some relevant features of the elements enclosed within the document. If, for instance, one simply indent all tags in a given document according to their nesting level, the sequence of indentation marks, as they appear within the document rotated by 90 degrees, can be looked at as a time series, whose shape roughly describe the document structure. These time-series data are then analyzed through their Discrete Fourier Transform (DFT), leading to abstract from structural details which should not affect the similarity estimation (such as different number of occurrences of an element or small shift in its position). More precisely, during a preorder visit of the XML document tree, as soon as a node is visited an impulse is emitted containing the information relevant to the tag. Thus: (1) each element is encoded as a real value; (2) the substructures in the documents are encoded using different signal shapes; (3) context information can be used to encode both basic elements and substructures, so that the analysis can be tuned to handle in a different way mismatches occurring at different hierarchical levels.

Once having represented each document as a signal, document shapes are analyzed through DFT. Some useful properties of this transform, namely, the concentration of the energy into few frequency coefficients, its invariance of the amplitude under shifts,

allow to reveal much about the distribution and relevance of signal frequencies without the need of resorting to edit distance based algorithms, and, thus, more efficiently. As the encoding guarantees that each relevant subsequence is associated with a group of frequency components, the comparison of their magnitudes allows the detection of similarities and differences between documents. With variable-length sequences, however, the computation of the DFT should be forced on M fixed frequencies, where M is at least as large as the document sizes, otherwise the frequency coefficients may not correspond. To avoid increasing the complexity of the overall approach, the missing coefficients are interpolated starting from the available ones. The distance between documents D_1 and D_2 is then defined as:

$$Dist(D_1, D_2) = \left(\sum_{k=1}^{M/2} (|[DFT^{\approx}(enc(D_1))](k)| - |[DFT^{\approx}(enc(D_1))](k)|)^2 \right)^{1/2}$$

where *enc* is the document encoding function, $DFT^{\approx}$ denotes the interpolation of DFT to the frequencies appearing in both D_1 and D_2, and M is the total number of points appearing in the interpolation. Comparing two documents using this technique costs $O(n \log n)$, where $n = max(|D_1|, |D_2|)$ is the maximum number of tags in the documents. The authors claim their approach is practically effective as those based on tree edit distance.

Link-based similarity. Similarity among documents can be measured relying on the links. Links can be specified at element granularity through ID/IDREF(S) attributes, or at document granularity through XLink specifications. To the best of our knowledge no link-based similarity measures have been specified tailored for XML documents at element granularity. At this granularity a measure should also consider the structure and content of the linked elements in order to be effective.

The problem of computing link-based similarity at document granularity has been investigated both for clustering together similar XML documents [Catania and Maddalena, 2002] and for XML document visualization as a graph partitioning problem . An XML document can be connected to other documents by means of XLink specifications both internal or externals. A weight can be associated with the link depending on a variety of factors (e.g., the type of link, the frequency it is used, its semantics). The similarity between two documents can be expressed in terms the weight of the minimum path between two nodes. Given a connection graph $G = (V, E)$ where each v in V represents an XML document, and each (v_1, v_2, w) is a direct w-weighted edge in E, [Catania and Maddalena, 2002] specifies the similarity between documents D_1 and D_2 as

$$Sim(D_1, D_2) = \begin{cases} 1 - \frac{1}{2^{cost(minPath(D_1, D_2)) + cost(minPath(D_2, D_1))}} & \text{if } existPath(D_i, D_j) = true, i, j \in 1, 2 \\ 0 & \text{otherwise} \end{cases}$$

Where, $e = minPath(v_1, v_2)$ is the minimal path e from v_1 to v_2, $cost(e)$ is the sum of the weights on the edge in e, $existPath(v_1, v_2) = true$ if a path exists from v_1 to v_2. A key feature of their approach is assigning a different weight to edges depending on the type of links (simple/extended, on load/on demand).

2.4 Characterization of heterogeneity

The objective of this section is to provide a precise characterization of what *heterogeneity* means in the context of XML collections. Such a characterization will allow us to compare collections in order to find out not only which collection is "more heterogeneous" than another one, but also to determine clusters of collections which are "heterogeneous in a similar way", and can be processed using similar techniques.

Unfortunately, while the concept of heterogeneity is used pervasively in the literature, it is seldom defined; an usual approach is to define an homogeneous collection as "collection that follows a single DTD or XML schema" [Theobald and Weikum, 2002]; unfortunately, this definition is not useful for our purposes, since DTDs (and XML Schema) allow for a large deal of diversity.

In addition, the term "heterogeneity" is used to refer to many different aspects of XML collections. Usually, these aspects are broadly classified into *structural heterogeneity* and *semantic heterogeneity*, a distinction inherited from the data integration literature. As an example of an XML collection with some structural heterogeneity, consider an XML-based bibliographic database, where entries for different types of publication (a book, an article, and so on) have a slightly different structure. In contrast, semantic heterogeneity is used to refer to cases where vocabulary is not strictly controlled, or natural language ambiguity is an issue. But such broad distinctions are difficult to substantiate. For instance, variations in tag names may be considered *both* structural (since the structure of two documents using different tag sets will necessarily be different) *and* semantic (since the difference may be treated as a vocabulary discrepancy).

In this chapter we will contribute quantitative criteria that can be used independently of a-priori classifications, and apply them to the study of XML collections. Section 2.4.1 formalizes the concept of heterogeneity in terms of population diversity. Section 2.4.2 applies these ideas to XML collections, in order to find a classification of collections in terms of their characteristics of heterogeneity. Finally, Section 2.4.3 reviews a series of approaches found in the literature in terms of the classification defined in the previous section.

2.4.1 Heterogeneity as diversity

We propose relating the ill-defined concept of heterogeneity to the idea of *diversity*, which is fundamental in the statistical study of populations (in a very broad sense) and has

been studied extensively. The classic definition of diversity is discussed by [Teachman, 1980]:

> We define *population diversity* to be the distribution of population elements (which are not limited to humans or the characteristics of humans) along a continuum of homogeneity to heterogeneity with respect to one or more variables.

The meaning of this definition is better explained by considering the properties that any measure of diversity should have. Assuming that N "population elements" are divided into I categories, and the proportion of elements falling into the i-th category is p_i with $\sum_i p_i = 1$:

- If any $p_i = 1$, i.e., if all events fall into one category, the measure should take a value of zero, indicating no variation.
- If all $p_i = 1/I$, i.e., if population elements are equally distributed among all categories, the measure should reach its theoretical maximum. Additionally, it would be desirable to normalize such a measure so that it equals 1 in this case.
- Postulating the existence of $I + 1$, $I + 2$ and so forth, additional categories to which no population members belong should leave the measures unaffected. In other words, given two populations with elements evenly distributed across all categories, the population with the greater number of categories should have the higher diversity. Conversely, combining two or more categories should not increase the measured diversity.

At least two measures of qualitatively variation possess these properties: *Simpson's D* [Simpson, 1949] and *Shannon's Entropy* [Shannon, 1948].

Simpson's D Simpson's D is defined as

$$D = 1 - \sum p_i^2$$

and can be normalized as

$$D_z = 1 - \frac{\sum p_i^2}{1 - \frac{1}{I}}$$

Simpson's D has been used extensively in the social sciences literature. One particularly interesting property is its interpretability: D is the probability that two population elements picked randomly from a population come from the same category.

Shannon's Entropy The *information (*or *Shannon) entropy* of a discrete random variable X that can take on possible values $x_1 ... x_n$ is

$$\begin{aligned} H(X) = E(I(X)) &= \sum_{i=1}^{n} p(x_i) \log_2 (1/p(x_i)) \\ &= -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \end{aligned}$$

where $I(X)$ is the information content or self-information of X, which is itself a random variable; and $p(x_i) = \Pr(X = x_i)$ is the probability mass function of X.

Information entropy quantifies the amount of information contained in a random variable; alternatively, it is a measure of the average information content we are missing when we do not know the value of the random variable.

It can be shown using the Jensen's inequality that

$$H(X) = E\left[\log_2 \left(\frac{1}{p(X)}\right)\right] \leq -\log_2 \left(E\left[p(X)\right]\right) \leq \log_2 n$$

We can use this formula to obtain a *normalized* version of entropy, H_0, whose values always lies between 0 and 1:

$$H_0(X) = \frac{H(X)}{\log_2 n}$$

Choosing a measure of diversity Many other measures of diversity have been proposed; for instance, Shannon's entropy is just one of a number of entropy indexes [Salicrú et al., 2005]. The choice of a particular measure depends largely of the application, but some comparative studies have been produced. We follow [Mcdonald and Dimmick, 2003], who study 13 measures of diversity in three different categories (probability-based, logarithm-based and rank-based), concluding that (*i*) overall, Shannon's Entropy and Simpson's D are indeed superior to other alternatives found in the literature, and (*ii*) there is a high correlation between different measures[1].

Given these results we will focus on entropy, since it has been much more widely used and studied in our community. Nevertheless, the techniques that we will present in the following sections are largely independent of a particular choice of diversity measure.

2.4.2 Entropy-based characterization of heterogeneous collections

In order to use the entropy measure on XML collections, we need to define which are the population elements of interest, and which categories will be used to measure diversity. Let us consider a simple example: the entropy of an XML collection based on the

[1] [Mcdonald and Dimmick, 2003] reports some advantages of entropy when sample sizes are small, but this is not particularly relevant in our context.

distribution of paths in the documents. Let $pathSet(C)$ be the set of all unique paths in collection C and $docSet(C)$ the set of documents in C. We define:

- The "population elements" are the individual paths in $pathSet(C)$
- For each path $\pi \in pathSet(C)$, let $docs(\pi)$ the set of documents in which π occurs. Thus, the "categories" will be the set $D = \bigcup docs(\pi_i)$ for all $\pi_i \in pathSet(C)$.

Let's see how entropy-based diversity is related to our intuitive concept of heterogeneity by studying the two extreme cases:

- In the most "homogeneous" case all documents are identical (ignoring ordering), and therefore paths appear in all documents.
- In the most "heterogeneous" case no document shares a path with any other documents, so every path appears in just one document.

In the first case, all paths belong to the same category, and therefore the entropy is 0. In the second case, the probability that one path belongs in a given category is exactly $1/|D|$, and therefore the entropy is maximal.

Different measures of entropy can be obtained in a similar way, by studying the probability of containment of one kind of objects (such as paths as in the previous example) into objects of a coarser granularity (such as documents). To be able to to this, we need to introduce a suitable definition of the granularities in an XML collection

Granularities in a collection Consider the granularity levels discussed in section 2.3.2.1. Let $\mathcal{DOC}$ be a set of XML documents, and $G = \{\texttt{DOC}, \texttt{ELEMENT}, \texttt{ATTRIBUTE}, \texttt{PATH}, \texttt{REGION}, \texttt{CONTENT}, \texttt{LINK}\}$ the granularity levels at which documents in $\mathcal{DOC}$ can be compared. Given a granularity level $\gamma \in G$, a mapping function m_γ can be defined for extracting from a collection $S \subseteq \mathcal{DOC}$ the portions of documents at that granularity level. For example, the mapping functions $m_{\texttt{ELEMENT}}$ and $m_{\texttt{PATH}}$ applied on a collection $S \subseteq \mathcal{DOC}$ return the set of elements (denoted $m_{\texttt{ELEMENT}}(S)$) and paths (denoted $m_{\texttt{PATH}}(S)$) occurring in S, respectively. A partial order relation $\prec_G$ between granularity levels of G can be defined, representing the containment relationship between granularity levels; it can be read as "is lower-level than". For instance, $\texttt{ELEMENT} \prec_G \texttt{PATH}$ because the paths in $m_{\texttt{PATH}}(S)$ are defined in terms of elements in $m_{\texttt{ELEMENT}}(S)$.

This granularity framework allows us to define a family of basic entropy measures of a collection, which we can derive directly from the $\prec_G$ binary relation between granularity levels. Namely, each pair of granularity levels related by $\prec_G$ induces a potentially useful entropy measure for an XML collection. Some examples are the probability of occurrence of a word within an element or a document; or the probability that an attribute will occur in a subtree.

Applications The entropy-based measures provide us with a method to characterize and compare collections of XML documents. As an example, we will study a set of rather diverse publicly available XML collections from the University of Washington XML Data Repository [2] and the INEX 2005 Heterogeneous track, most of which are used for experimentation in the literature. Tables 2.4 and 2.5 provide a description of the collections involved in each of these sources.

The first application of entropy-based characterization is comparing the heterogeneity. For instance, consider the following entropy-based variables:

1. *path*: diversity of *paths* in the collection
2. *tag*: diversity of *tags* in the collection
3. *word*: diversity of *words* in the collection
4. *attr*: diversity of *attributes* in the collection
5. *attrword*: diversity of *words in the attributes* in the collection

We have avoided defining more specific categories, since many of these collection consist of just a single large XML document. Thus, these variables are broadly applicable to many collections. Figure 2.13 shows the values of the entropy of these variables in two bibliographic collections, displayed as radial plots. This allows us to easily compare the different characteristics of the collections: in this case, both collections display similar textual complexity (in both cases, textual content consists mainly of author names, article titles and abstracts), but one collection shows more structural diversity (*tag* and *path* variables), and the attribute-related characteristics are completely different. This information can be useful in selecting indexing and processing techniques for the collections.

We can generalize this approach and cluster all collections into groups that share similar characteristics of heterogeneity. For this example, we will limit ourselves to the *path*, and *word* variables, since many collections have no information encoded in attributes[3]. Considering the values of the variables as vectors of characteristics, we used a PAM (partitioning around medoids) algorithm to find the clustering show in in Figure 2.14. The number of clusters (4) was estimated using the silhouette width criterion [Rousseeuw, 1987].

The groupings in Figure 2.14 can be interpreted as follows:

- Cluster 1 contains collection with relatively low structural and high textual heterogeneity. Most collections in this cluster consist of relational data converted into

[2] `http://www.cs.washington.edu/research/xmldatasets/`

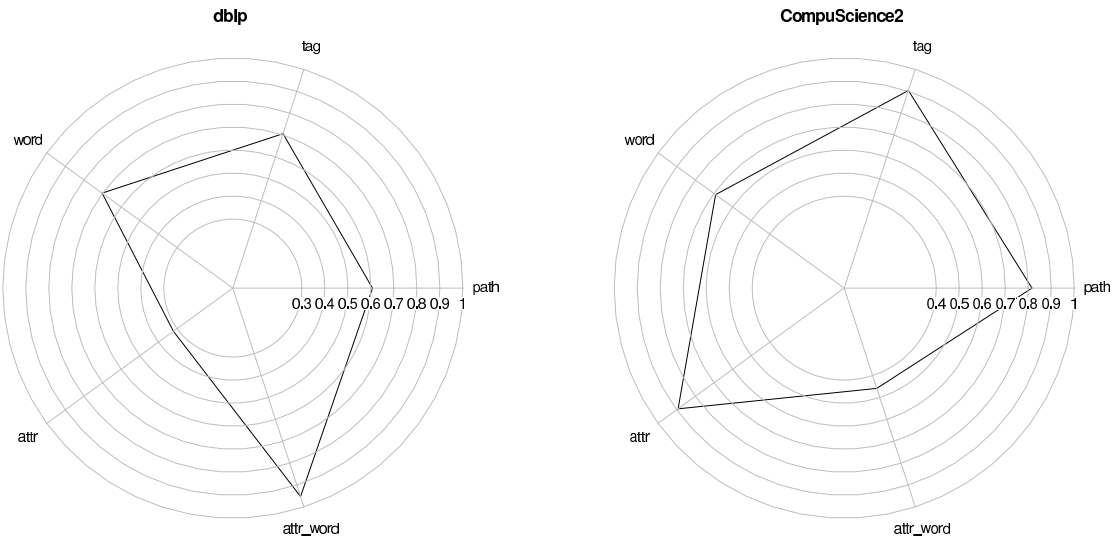
[3] Variable *tag* is not included, since an information-theoretic analysis shows that it is redundant when variable *path* is also considered.

Name	Description	Size
Protein Sequence Database (psd7003)	Integrated collection of functionally annotated protein sequences.	683 MB
SwissProt	SWISS-PROT is a curated protein sequence database which strives to provide a high level of annotations (such as the description of the function of a protein, its domains structure, post-translational modifications, variants, etc.), a minimal level of redundancy and high level of integration with other databases.	109 MB
Auction Data (321gone, ebay, ubid, yahoo)	Auction data converted to XML from web sources.	23 KB
DBLP Computer Science Bibliography (dblp)	The DBLP server provides bibliographic information on major computer science journals and proceedings. DBLP stands for Digital Bibliography Library Project.	127 MB
University Courses (reed, uwm, wsu)	Course data derived from university websites.	277 KB
NASA	Data sets converted from legacy flat-file format into XML and made available to the public.	23 MB
SIGMOD Record	Index of articles from SIGMOD Record	467 KB
TPC-H Relational Database Benchmark (part, lineitem, partsupp, supplier, orders, nation, region, customer)	TPC-H Benchmark, 10 MB version, in XML form.	603 KB
Treebank	English sentences, tagged with parts of speech. The text nodes have been encrypted because they are copyrighted text from the Wall Street Journal. Nevertheless, the deep recursive structure of this data makes it an interesting case for experiments.	82 MB
Mondial	World geographic database integrated from the CIA World Factbook, the International Atlas, and the TERRA database among other sources.	1 MB

Table 2.4: Collections in the University of Washington XML Data Repository. The short names between parenthesis are used to refer to the collections in the experiments of this section.

Name	Size
Berkeley2	5 MB
bibdbpub2	579 KB
CompuScience2	80 MB
dblp2	18 MB
hcibib2	8 MB
qmuldcsdbpub2	241 KB
SIGMOD Record	80 KB
zdnet-xml	39 MB

Table 2.5: INEX 2005 Heterogeneous track collections. All of these collection consist of bibliographic data.



(a) DBLP Collection

(b) The *Collection of Computer Science Bibliographies* collection

Figure 2.13: Radial plots comparing the heterogeneity of the dblp and CompuScience2 collection using five entropy-based variables

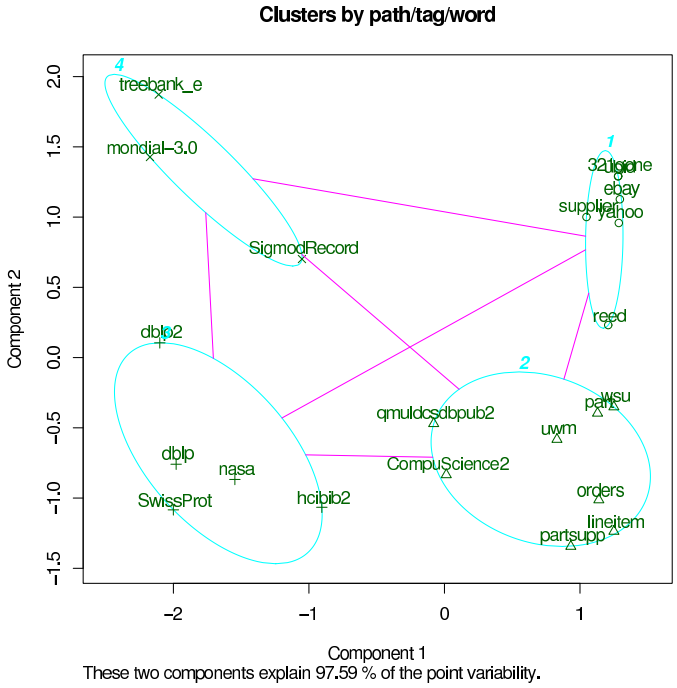


Figure 2.14: Clustering collections by heterogeneity characteristics

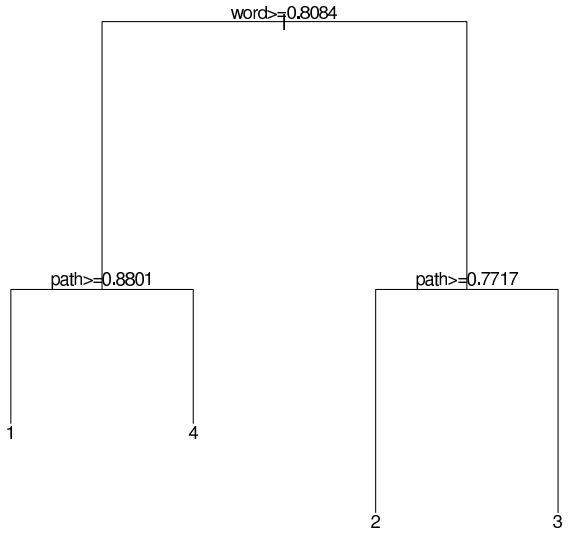


Figure 2.15: Regression tree for membership in collection clusters

XML; in these collections we find very little structural variation, and all information is effectively encoded in the textual content.

- Cluster 2 contains collection with relatively low structural and textual heterogeneity.
- Cluster 3 contains collection with relatively high structural and low textual heterogeneity.
- Cluster 4 contains collection with relatively high structural and textual heterogeneity.

These results are summarized by the regression tree displayed in Figure 2.15.

The collections presented so far represent the "mainstream" in XML data management, since most refinements to existing techniques are tested using these collections, or collections with similar characteristics, as we shall show in Section 2.4.3. But there are other applications which require handling XML collections, in which standard XML techniques are not considered to be applicable because the data are too complex or too heterogeneous. As an example, we will consider two more collections: the Assam data set[4], which consists of a set of conceptual description of Web Services, and a collection of documents encoded in LandXML[5], a standard in Geographical Information Systems applications.

[4]http://moguntia.ucd.ie/repository/datasets/ `http://moguntia.ucd.ie/repository/datasets/`
[5]http://www.landxml.org/ `http://www.landxml.org/`

Figure 2.16: Clustering collections by heterogeneity characteristics, including two highly heterogeneous collections.

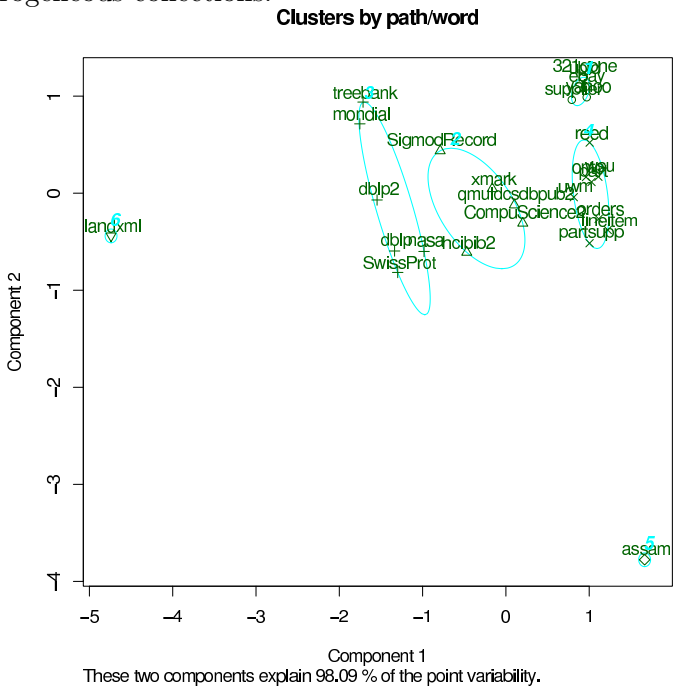


Figure 2.16 shows how the addition of these two new collections leave the original categories unchanged (except for some minor adjustments in boundary cases), but reveals a large space of highly heterogeneous collections which is not covered by traditional XML techniques.

2.4.3 Characterization of retrieval approaches with respect to heterogeneity

We will finish our discussion by characterizing a number of relevant approaches found in the literature, including some that explicitly address "heterogeneous" data, including text-oriented (XRank [Guo et al., 2003]), probabilistic (TopX [Theobald, 2006] and XXL [Theobald and Weikum, 2002]) and relaxation-based ([Amer-Yahia et al., 2002], [Amer-Yahia et al., 2005]). Table 2.6 describes the collections used by the authors to showcase their approaches.

The table shows that all of the collections used belong in clusters 2 and 3, and only [Amer-Yahia et al., 2005] deals with a more challenging collection in cluster 4.

Approach	Collections used	Other collections
XRank	DBLP, XMark	
TopX	INEX	IMDB,TREC Terabyte
XXL	SIGMOD	Religious books, Shakespeare plays, synthetic bibliographies
[Amer-Yahia et al., 2002]	DBLP	
[Amer-Yahia et al., 2005]	Treebank	

Table 2.6: A comparison of collections used in in the literature

2.5 Concluding remarks

This chapter has presented the basic results needed to put the contributions of this thesis into context. After presenting an introduction to the main developments in the area, a survey of the literature provides support for one key observation: many approximate approaches reuse a small set of basic similarity measures, with the necessary tailoring to the problem at hand. Finally, we have presented a formal characterization of heterogeneity based on information-theoretic considerations, which is broadly useful to many different XML applications.

Chapter 3

A multi-similarity framework for XML

This chapter presents a framework to create flexible similarity measures for XML by combining existing general-purpose similarity functions. First, Section 3.1 introduces our approach and describes related work. Section 3.2 presents the basic concepts necessary for the combination of independent measures, by applying the *composition* design pattern. Then, Section 3.3 develops a method to describe measure metadata using Description Logic; this allows applications to generate, store and process measure metadata using standard OWL-based tools, which provide many useful primitives, in particular the possibility to verify the consistency of a complex measure. Finally, Section 3.4 revisits the main XML similarity-based techniques, and determines which approaches can be readily adapted into generic components that can be integrated into a multi-similarity application.

3.1 Introduction and related work

Section 2.3 has shown that a wide variety of similarity measures, both general purpose and specifically tailored, has been proposed for XML [Guerrini et al., 2006]. Each measure produces good results when the collections present specific characteristics and are hardly reusable in other collections. General purpose measures include metric functions such as the Manhattan or Euclidean distances; IR-like matching coefficients such as the cosine with $tf \times idf$ weighting; entropy-based measures such as the Kullback-Leibler divergence; and structure-oriented techniques such as variants of the Tree Edit Distance algorithm [Selkow, 1977].

These similarity measures are usually obtained through the composition of several "atomic" measures at a given *granularity level* of the XML hierarchy. For instance, a measure for complete XML documents is defined in terms of paths similarity, which in turn requires some criterion to compare the elements in the path. The granularity levels introduced in Section 2.4.2 that can be considered are: the whole XML document, subtrees (i.e., regions of documents), paths, elements, links, attributes and textual content (of attributes and data content elements).

This indicates that it is possible to build frameworks for the implementation of complex XML similarity measures, based on a library of basic component functions (implementing

the atomic measures). This does not exclude the employment of ad-hoc measures, if necessary. For example, the designer of a similarity-based application in the domain of genetics may need to combine a generic text-oriented function that matches protein names with a highly specialized function that matches amino acid sequences.

Several works have presented approaches to treat diverse similarity measures in a generic, reusable way. The work that introduced the concept of multi-similarity, [Adalı et al., 1998], uses the concept of *similarity abstraction*, which encapsulates the notion of an arbitrary similarity function which can be manipulated independently of the underlying implementation. Ontology alignment systems such as OLA [Euzenat et al., 2005] and COMA [Do and Rahm, 2002] support the combination of different measures. COMA++ , the successor system to COMA, is of particular interest, since it includes the notion of combining measures at different granularity levels. A downside of these approaches is that they are focused on the integration of small-scale schemas (SQL, XSD) and OWL-encoded ontologies which can be manipulated using memory-based algorithms, and are therefore not suitable for use in large-scale XML collections.

A crucial feature in any system that supports true multi-similarity is that it should be possible to add new measures dynamically into the system. This requires the existence of a metadata repository, which keeps information about the existing measures and its characteristics. In this respect, OLA uses a hard coded set of measures, while COMA includes a RDBMS-based onternal store. As an extension of this concept, we propose coding the relevant metadata using a Description Logic, which can be encoded using the standard OWL language. The main advantage of this approach is the availability of tools for metadata management, whose features include facilities for creating semantically rich descriptions which can be processed, and checked for consistency, using reasoners.

3.2 Measure components

This section defines a formal framework for the definition and composition of similarity measures, relying on the granularity levels of XML documents. and then specified software components implementing such functions that can be combined to obtain new measures specific for a given context.

3.2.1 A Formal Specification of Similarity Measures

Consider the granularity levels in an XML collection, as defined in Section 2.4.2. A similarity measure at a given granularity level γ can be defined as a function $f_\gamma : m_\gamma(S) \times m_\gamma(S) \rightarrow [0, 1]$. Moreover, it can be expressed in terms of other lower level similarity functions. For example, if we denote elements by e_n and paths by p_m, then an instance of similarity function for paths expressed in terms of a similarity function for elements is

$$f_{\text{PATH}}(p_1,p_2) = \frac{\sum_{i,j} f_{\text{ELEMENT}}(e_i, e_j)}{|p_1||p_2|}$$

where in the simplest case $f_{\text{ELEMENT}}(e_i, e_j) = 1$ if $e_i = e_j$ and 0 otherwise, and $|p|$ denotes the length of path p.

3.2.2 Composition of measures

The basic unit of composition is the *measure component.* A *measure component* is an implementation of a similarity function at a given granularity level. It can depend on one or more lower-level components, but it is irrelevant *which* concrete lower-level component is used, as long as it belongs to the right granularity level. This capability of creating complex measures by combining simpler parts which are measures themselves is analogous to the *Composite* pattern commonly used in Software Engineering [Gamma et al., 1995]; the general model for measure composition can be illustrated by the UML diagram in Fig. 3.1. In addition, every component can be parametrized; for instance, a component that computes similarity at the textual level may allow the user to choose whether common words ("stop words") must be considered.

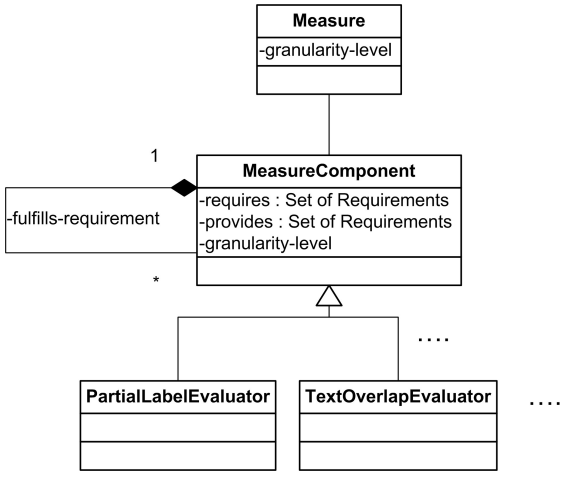


Figure 3.1: Simplified UML model describing the application of the *Composite* pattern to the creation of complex measures: A *MeasureComponent*'s requirements are fulfilled by other *MeasureComponent*s, creating the provides/requires hierarchy.

Using this framework, a large number of functions with different requirements can be defined for each granularity level. In order to characterize them, the partial ordering of levels $\prec_G$ is extended into a *provides/requires hierarchy* typically used in software

component engineering [Plašil and Visnovsky, 2002]. Each measure component is thus tagged with two extra properties, whose values are chosen from a predefined set of *features*: the *provides* property indicates the granularity level at which the function operates (e.g., all node-level functions provide the feature *nodeMatch*), while the *requires* property indicates the features that must be provided by the lower-level functions on which the function relies.

For instance, let $C_{\texttt{PATH}}$ be a component that implements the $f_{\texttt{PATH}}$ function defined above. Then, $provides(C_{\texttt{PATH}}) = \{pathMatch\}$ and $requires(C_{\texttt{PATH}}) = \{elementMatch\}$. For the component to be usable, another component that provides $elementMatch$ must be available.

The provides/requires hierarchy separates the representation of the components from the actual similarity function implemented. This allows us to implement components for generic operations (usually called "tie components" [Plašil and Visnovsky, 2002]) by computing an aggregated value out of the results of other components. For instance, consider the "weighted sum" tie component $C_{\texttt{WSUM}}$, that can be defined for a set of n components $\{C_1, \ldots, C_n\}$ at the same granularity level γ and a set of n weights $\{w_1 \ldots w_n | w_i \in \mathbb{R}\}$:

$$\begin{aligned} parameters(C_{\texttt{WSUM}}) = &\quad \{\{C_1 \ldots C_n\}, \{w_1 \ldots w_n\}\} \\ provides(C_{\texttt{WSUM}}) = &\quad \textstyle\bigcap_i provides(C_i) \\ requires(C_{\texttt{WSUM}}) = &\quad \textstyle\bigcup_i requires(C_i) \\ f_{C_{\texttt{WSUM}}} : &\quad G \times G \rightarrow [0,1] \\ f_{C_{\texttt{WSUM}}}(o_1,, o_2) = &\quad \displaystyle\sum_i w_i \times f_{C_i}(o_1, o_2) \end{aligned}$$

Fig. 3.2 shows an instantiation of a multilevel similarity measure that uses *WeightedSum*, the implementation of $C_{\texttt{WSUM}}$. It combines components at the node, label, and text granularity levels. Note how the components at the node similarity level are computed using a weighted sum. Edge labels represent the requires/provides hierarchy.

3.3 DL-based measure metadata

We have just shown how to create components that can be glued together to form flexible similarity measures. However, the data engineer needs to answer higher-level questions. For instance: given a particular component, which other components are available to fulfill its requirements? Or, is this component sound (i.e. all of its requirements are correctly fulfilled)? This is particularly important in collaborative methodologies, in which sharing components among independently-working engineers is crucial.

Our approach consists in encapsulating all of the required consistency rules in a declarative formalism, using a suitable Description Logic (DL) [Baader et al., 2003]. DLs

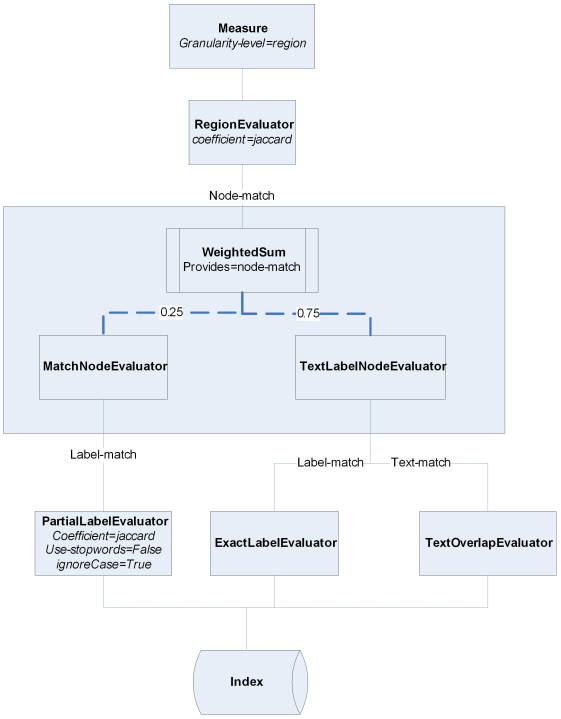


Figure 3.2: Component structure of a similarity measure

define *concepts* as sets of *individuals*, and *roles* as binary relationships between concepts. New concepts can be defined by enumeration of individuals, or using a series of primitives, which vary depending of the expressivity of the particular logic being used. The most basic primitives are *concept conjunction* ($\sqcap$), *concept disjunction* ($\sqcup$), and *complement of a concept* ($\neg$).

For example, consider the concepts Female, Person and Woman, and the roles hasChild and hasFemaleRelative[1]. We can create a new concept *Male* as "persons that are not female":

$$\mathsf{Male} \equiv \mathsf{Person} \sqcap \neg \mathsf{Female}$$

This new concept can be used in new definitions, for instance in "individuals that are either female or male":

$$\mathsf{Male} \sqcup \mathsf{Female}$$

Another important feature are *role restrictions*, which include existential and universal quantifications and cardinality restrictions. For example, the "individuals that have a

[1]Following [Baader et al., 2003], DL formulas are rendered using a Sans Serif font. Concepts are shown in UpperCamelCase and roles are shown in lowerCamelCase, and names of individuals are all uppercase.

female child" are described by the expression $\exists$hasChild.Female, while the "individuals all of whose children are female" by the concept expression $\forall$hasChild.Female. In both cases, the second argument of the role is called the *role filler*. Cardinality restrictions are expressed by $\leq$,$\geq$and $=$ primitives; for example, the "individuals having at least three children and at most two female relatives" are described as:

$$(> 3\text{hasChild}) \sqcap (\leq 2\text{hasFemaleRelative})$$

Description Logics are specially well suited for the modeling of metadata, and have the advantage that they support a variety of reasoning tasks (subsumption, instance checking, relation checking, concept consistency and knowledge base consistency) which can be exploited in our context to automate many of the tasks that must be performed by the data engineer when designing a multi-similarity system. In addition, sufficiently expressive DLs provide "inverse functional" roles, which are exactly equivalent to candidate keys in a database [De Giacomo and Lenzerini, 1994]. This is useful to support the semi-automatic specification of indexes as a combination of index components.

In the remainder of this section we develop a mapping of the concepts we have just described to standard Description Logic notation. This will provide us with a formal framework for the representation and manipulation of metadata about measures.

The first thing that is needed is a vocabulary for translating the composition pattern. We introduce the generic transitive role hasPart and its inverse isPartOf $\equiv$ hasPart^{-}. This partOf/isPartOf pair of roles suffices for a simple description of composite concepts. The most basic possible definition of a composite is :

$$\exists\text{hasPart.Composite} \sqcap \forall\text{hasPart.Composite}$$

which corresponds to the set of concepts as the set of individuals having at least one filler of the role partOf belonging to the concept Composite.

By refining this definition we can obtain a description of a generic component:

$$\begin{aligned} \text{Component} \equiv\ & \exists\text{hasPart.Component} \\ & \sqcap \forall\text{hasPart.Component} \\ & \sqcap = 1\text{hasGranularity.GranularityLevel} \\ & \sqcap \forall\text{hasGranularity.GranularityLevel} \\ & \sqcap \exists\text{provides.Requirement} \\ & \sqcap \exists\text{requires.Requirement} \end{aligned}$$

which introduces the concept of a *GranularityLevel*, of which a Component must have exactly one. For the purposes of measure description, we can define it as a simple enumeration of individuals:

$$\mathsf{GranularityLevel} \equiv \{\mathsf{DOCUMENT}, \mathsf{REGION}, \mathsf{PATH}, \mathsf{ELEMENT}, \mathsf{LINK}, \mathsf{ATTRIBUTE}, \mathsf{TEXT}\}$$

though, if necessary, this definition can be easily modified to model the relations between granularity levels which are depicted in Figure 2.6.

The provides/requires hierarchy is defined analogously, using the roles provides and requires and the concept Requirement.

Example 1 The RegionEvaluator component depicted in Figure 3.2 can be described as:

$$\begin{aligned}\mathsf{RegionEvaluator} \equiv\ & \mathsf{Component} \\ & \sqcap \exists \mathsf{hasPart}.(\mathsf{Component} \sqcap \mathsf{hasGranularity.ELEMENT}) \\ & \sqcap \forall \mathsf{hasGranularity.REGION} \\ & \sqcap \exists \mathsf{provides.RegionMatchRequirement} \\ & \sqcap \exists \mathsf{requires.NodeMatchRequirement} \\ & \sqcap\ = 1 \mathsf{hasCoefficient.SimCoefficient}\end{aligned}$$

where the concept SimCoefficient represents the set of available similarity coefficients, e.g.

$$\mathsf{SimCoefficient} \equiv \{\mathsf{JACCARD}, \mathsf{DICE}, \mathsf{OVERLAP}\}$$

and RegionMatchRequirement and NodeMatchRequirement are specializations of Requirement[2].

An equivalent (and far more verbose) representation of this definition in the OWL markup language is depicted in Figure 3.3. ○

The availability of representations in the standard OWL language makes it possible to take advantage of existing tools, including editors, reasoners and libraries. This greatly facilitates the integration of DL-based description in real applications.

3.4 Re-usability of existing approaches

Most of the similarity-based techniques for XML described in the previous chapter incorporate different measures for different similarity levels, and in some cases they allow some flexibility in the selection of a particular measure at a particular granularity level. However, instead of the modular approach presented in this paper, they are conceived

[2] Ideally, the provides/requires hierarchy should be inferred from hasPart/isPartOf relationships. Unfortunately, due to some limitations in the handling of rules as provided by currently available DL management tools it is more convenient to just assert the values into the A-Box.

```
<owl:Class rdf:ID="RegionEvaluator">
    <rdfs:subClassOf rdf:resource="#Component"/>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasPart"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:intersectionOf rdf:parseType="Collection">
                        <owl:Class rdf:about="#Component"/>
                        <owl:Restriction>
                            <owl:onProperty rdf:resource="#hasGranularity"/>
                            <owl:hasValue rdf:resource="#Element"/>
                        </owl:Restriction>
                    </owl:intersectionOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasGranularity"/>
            <owl:hasValue rdf:resource="#Region"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#provides"/>
            <owl:someValuesFrom rdf:resource="#RegionMatchRequirement"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#requires"/>
            <owl:someValuesFrom rdf:resource="#NodeMatchRequirement"/>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Class>
            <owl:intersectionOf rdf:parseType="Collection">
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasCoefficient"/>
                    <owl:cardinality rdf:datatype="&xsd;int">1</owl:cardinality>
                </owl:Restriction>
                <owl:Restriction>
                    <owl:onProperty rdf:resource="#hasCoefficient"/>
                    <owl:allValuesFrom rdf:resource="#SimCoefficient"/>
                </owl:Restriction>
            </owl:intersectionOf>
        </owl:Class>
    </rdfs:subClassOf>
</owl:Class>
```

Figure 3.3: OWL representation of component RegionEvaluator

as complex monolithic functions. In most cases, however, these flexible parts can be easily refactored and converted into reusable compositional measures. Conversely, these approaches would be enriched by allowing the rich parametrization permitted by the model presented in this chapter.

In this section we revisit the approximate XML approaches described in Section 2.3.1, analyzing which parts in each technique are amenable to being transformed into generic components and reused in different contexts.

First of all, ELIXIR [Chinenyanga and Kushmerick, 2002] is based on a textual measure based on $tf \times ifd$. It is worthy to note that this measure is literally plugged into the system, as it is implemented by an external systems (WHIRL [Cohen, 1998]). In contrast, XIRQL [Fuhr and Großjohann, 2001] makes a provision for flexible, user-defined concepts of similarity at the textual level:

> For each data type, there is a special operator symbol. This enables users to use string equality even on date values, if desired. It also obviates the need for type inference, which would open a can of worms. One operator, `$c$`, has already been presented which implements a 'contains' operation for natural language text. As another example, for person names, a useful operator would be `$sounds_like$`.

XXL [Theobald and Weikum, 2002] proposes an ontology-based similarity measure for terms and element names, which uses semantic relations (hypernymy, homonymy. and so on). It also provides a node similarity measure based on structural relations. [Damiani and Tanca, 2000] rely on "fuzzy weights" to express the relative importance of tags and subtrees, ans make a provision for "user-defined query semantics". Finally, [Amer-Yahia et al., 2005] defines $tf \times idf$-based measures for elements, paths and twigs (XML subtrees).

The only approaches that present difficulties in refactoring are [Amer-Yahia et al., 2002] and [Schlieder, 2001]. Nevertheless, these approaches can be incorporated into our framework as monolithic components at the subtree level.

All of these potentially reusable components have been described using the base concepts techniques presented in the previous section, forming a common component model. The components that have been incorporated are summarized in Table 3.1.

Example 2 Figure 3.4 shows the structure of a measure that reuses the XXL ontology-based measure for matching nodes, modified to use XIRQL's *contains* operation for textual content matching. ○

As mentioned above, the possibility to encode the component model using a standard language (OWL) makes it possible to benefit from existing tools. Figure 3.5 shows the OWL description of the components used to create the measure in the previous example,

	Subtree	Path	Element	Tag	Text
ELIXIR					✓
XIRQL					✓
XXL				✓	✓
[Damiani and Tanca, 2000]	✓			✓	
[Amer-Yahia et al., 2005]	✓	✓	✓		

Table 3.1: Potentially reusable components in existing approximate XML techniques

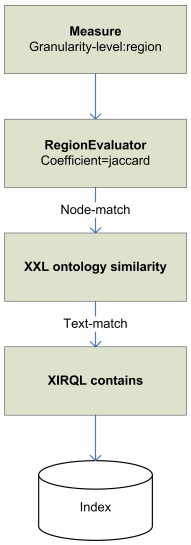


Figure 3.4: A similarity measure combining components extracted from XXL and XIRQL

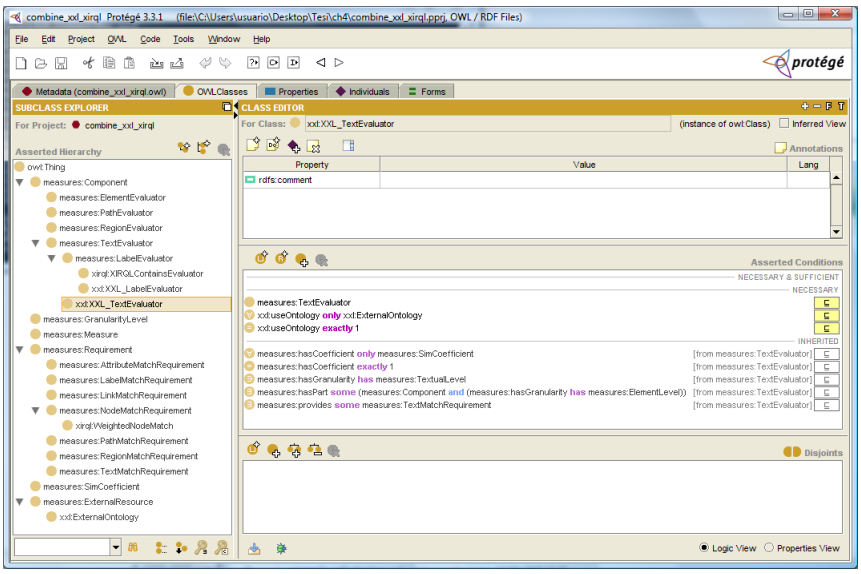


Figure 3.5: The Protégé view of the XXL textual similarity function. This model also includes the relevant XIRQL components required for the creation of the measure in Example 2

loaded into the Protégé editor[3], which is OWL-aware. Figure 3.6 shows the underlying OWL code.

The most important benefit of OWL encoding, however, is the possibility to check the consistency of measure models using DL reasoners. For concrete applications, standard off-the-self editors such as Protégé are too generic, and specialized tools are more convenient for particular applications. Chapter (6) presents a measure editor which is tailored to the editing of measure component in a multi-similarity context.

3.5 Summary

This chapter presents a model for highly customizable measures, suitable for use in multi-similarity applications. The model is based on the composition of basic components. A method to describe the measures using Description Logics is also presented, which provides several advantages due the ability to reuse OWL-based standard and tools; in particular, a standard machine-readable representation format, and the ability to use reasoners to check the consistency of a measure. Finally, existing XML similarity-oriented techniques are analyzed, in order to check which ones can be exploited to form generic, reusable measure components.

[3] `http://protege.stanford.edu/`

```
<owl:Class rdf:ID="XXL_TextEvaluator">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="useOntology"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#useOntology"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:ID="ExternalOntology"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://krono.act.uji.es/ns/measures.owl#TextEvaluator"/>
</owl:Class>
```

Figure 3.6: OWL description of the XXL text evaluator component, which depends on an external ontology for operation

Chapter 4

A Flexible pattern-based querying model for semistructured data

This chapter introduces a set of techniques for approximate querying of XML data. Section 4.1 motivates the chosen approach. Section 4.2 formally introduces the notion of pattern, target, fragments and region the approach relies on. Section 4.3 discusses how to efficiently identify fragments and regions, and Section 4.4 presents experimental results.

4.1 Introduction

Interoperability among systems is commonly achieved through the interchange of XML documents, that can represent a great variety of information resources: semi-structured data, database schemas, concept taxonomies, ontologies, etc. Most XML document collections are highly heterogeneous from several viewpoints. The first level of heterogeneity is *tag heterogeneity*: vocabulary discrepancies may occur in the element tags. Two elements representing the same information can be labeled by tags that are stems (e.g., `author` and `authors`), that are one substring of the other (e.g., `authors` and `co-authors`), or that are similar according to a given thesaurus (e.g., `author` and `writer`). The second level of heterogeneity is *structural heterogeneity* that results in documents with different hierarchical structures. Structural heterogeneity can be produced by the different schemas (i.e., DTDs or XML Schemas) behind the XML documents. Moreover, as schemas can also include optional, alternative, and complex components, structural heterogeneity can appear even for a single schema collection. In a structurally heterogeneous collection, a parent-children relationship (e.g., the `address` element child of the `person` element) can be relaxed into an ancestor-descendant relationship (e.g., the `address` element is a descendant of the `person` element) but it can also be inverted (e.g., the `person` element is a child/descendant of the `address` element) or the two elements can appear as siblings, or even combined in a single tag (`person_address`).

One of the most useful operation on XML documents is pattern-based retrieval: given a user-provided pattern all its occurrences in the collection must be retrieved (*subtree identification*) [Kilpeläinen, 1992]. Document heterogeneity poses several challenges to this retrieval. In the case of heterogeneous XML collections, indeed, this retrieval

must also be approximate, that is, document subtrees that best fit the user requests must be detected. The hierarchical structures of XML documents in the collection, indeed, may only partially conform to the hierarchical structure of the pattern. For these reasons, a user query could be answered by a set of subtrees presenting slight variations in their labels as well as in their structures. Existing approaches to approximate query answering for XML documents [Amer-Yahia et al., 2002, 2005, Marian et al., 2005, Schlieder, 2002] adapt query evaluation techniques to accommodate non-exact matching and top-k processing, by weakening some constraints in the user query. The approach most often adopted in literature is weakening the parent-children relationship to the ancestor-descendant relationship.

The techniques presented in this chapter stress the tag and structural heterogeneity of XML document collections. This can lead to search a very large amount of highly heterogeneous documents (the *target*). In this context, we propose an approach for identifying the portions of documents that are similar to a given *pattern*. A pattern is an abstract representation of a user request. Documents in the target collection may exhibit weak similarity to the pattern. Thus, with respect to the mentioned proposals, we take the reverse approach: rather than weakening strict structure-based retrieval, we completely disregard structure in the beginning. We thus develop a two-phase approach where, in the first phase, structural constraints expressed by the pattern are not taken into account in identifying the portions of the target in which the nodes of the pattern appear. The structural similarity between the pattern and the identified portions of the target is evaluated as a second step, allowing to rank the identified portions and producing the result. In this second phase, different similarity measures can be considered, thus accounting for different degrees of document heterogeneity, depending on the application domain and on the heterogeneity degree of the target collection.

The proposed approach is thus highly flexible. The problem is however how to perform the first step efficiently, that is, how to efficiently identify *fragments*, i.e., portions of the target containing labels similar to those of the pattern, without relying on strict structural constraints. Our approach employs ad-hoc data structures: a *semantic inverted index* of the target and a *pattern index* extracted from the inverted index on the basis of the pattern labels. Through the semantic inverted index, nodes in the target with labels similar to those of the pattern are identified and organized in the levels in which they appear in the target. Fragments are generated by considering the ancestor-descendant relationship among such vertices. Moreover, identified fragments are combined in *regions*, allowing for the occurrence of nodes with labels not appearing in the pattern, if the region exhibits a higher structural similarity with the pattern than the fragments it is originated from. Finally, some heuristics are employed to avoid considering all the possible ways of merging fragments into regions and for the efficient computation of similarity, thus making our approach more efficient without losing precision.

In this chapter, we formally define the notions of fragments and regions and propose the algorithms allowing their identification, relying on our pattern index structure. The

use of different structural similarity functions taking different structural constraints (e.g., ancestor-descendant and sibling order) into account is discussed. The practical applicability of the approach is finally demonstrated, both in terms of quality of the obtained results and in terms of space and time efficiency, through a comprehensive experimental evaluation. The contribution of the chapter can thus be summarized as follows: *(i)* specification of an approach for the efficient identification of regions by specifically tailored indexing structures; *(ii)* characterization of different similarity measures between a pattern and regions in a collection of heterogeneous tree structured data; *(iii)* characterization and experimental analysis of the different degrees of heterogeneity in XML document collections; *(iv)* thorough experimental validation of the approach.

4.2 Pattern, target, fragment, and region trees

In our approach, both patterns and targets are represented as trees. Thus, fragments and regions are (sub)trees as well. In this section, we introduce these notions. First of all, however, we introduce some basic notions and some useful notations on trees, together with some functions for handling element tag similarity.

4.2.1 Trees

The notation used throughout this paper to represent trees is fairly standard. A *tree* is a structure $T = (V, E)$, where V is a finite set of *vertices*, E is a binary relation on V that satisfies the following conditions: (i) the root (denoted $root(T)$) has no parent; (ii) every node of the tree except the root has exactly one parent; (iii) all nodes are reachable via edges from the root, that is, $(root(T), v) \in E^*$ for all nodes in V (E^* is the Klein closure of E). If $(u, v) \in E$, we say that (u, v) is an *edge* and that u is the *parent* of v (denoted $\mathcal{P}(v)$). A *labeled tree* is a tree with which a node labeling function *label* is associated. Given a tree $T = (V, E)$, Table 4.1 reports functions and symbols used throughout the paper. In the following, when using the notations, the tree T will not be explicitly reported whenever it is clear from the context. Otherwise, it will be marked as subscript of the function.

Node order is determined by a pre-order traversal of the document tree [Grust, 2002]. In a pre-order traversal, a tree node v is visited and assigned its pre-order rank $pre(v)$ before its children are recursively traversed from left to right. A post-order traversal is the dual of the pre-order traversal: a node v is visited and assigned its post-order rank $post(v)$ after all its children have been traversed from right to left. The *level* of a node v in the tree is defined, as usual, by stating that the level of the root is 1, and that the level of any other node is the successor of the level of its parent. The *position* of a node v among its siblings is defined as the left-to-right position at which v appears among the nodes whose parent is the parent of v. Each node v is coupled with a quadruple $(pre(v), post(v), level(v), pos(v))$ as shown in Figure 4.2.1 ($pre(v)$ is used as

Symbol	Meaning
$root(T)$	root of T
$\mathcal{V}(T)$	set of vertices of T (i.e., V)
$\|V\|$,$\|T\|$	cardinality of $\mathcal{V}(T)$
$label(v)$,$label(V)$	label associated with a node v and the nodes in V
$\mathcal{P}(v)$	parent of vertex v
$pre(v)$	pre-order traversal rank of v
$post(v)$	post-order traversal rank of v
$level(v)$	level in which v appears in T
$level(T)$	depth of T
$pos(v)$	left-to-right position at which v appears among its siblings
$desc(v)$	set of descendant of v ($desc(v) = \{u \mid (v,u) \in E^*\}$)
$nca(v,u)$	nearest common ancestor of v and u
$d(v)$	distance of node v from the root
d^{max}	maximal distance from the root ($d^{max} = max_{v \in \mathcal{V}(T)} d(v)$)

Table 4.1: Notations

node identifier). For the sake of clarity, in the following graphics we omit the quadruples when they are not relevant for the discussion.

Pre- and post-ranking can also be used to efficiently characterize the descendants u of v. A node u is a descendant of v, $v \in desc(u)$, iff $pre(v) < pre(u) \wedge post(u) < post(v)$. There are some other useful properties that are trivially computed using these numbers. If, given two nodes, one is neither an ascendant nor a descendant of the other, then they are either left- or right-relatives. A node v is a *left-relative* of a node u if $pre(v) < pre(u)$ and $post(v) < post(u)$. The definition of *right-relative* is analogous. Given a tree $T = (V, E)$ and two nodes $u, v \in V$, the *nearest common ancestor* of u and v, $nca(u, v)$, is the common ancestor of u and v such that any other common ancestor of u and v is an ancestor of $nca(u, v)$. Note that $nca(u, v) = nca(v, u)$, and $nca(u, v) = v$ if u is a descendant of v. The distance $d(v)$ of a node v from the root, which coincides with its pre-order rank, amounts to the number of nodes traversed in moving from the root to the node in the pre-order traversal. The maximal distance d^{max} corresponds to the number of nodes in the tree, i.e., $|T|$. d^{max} also corresponds to the post-order rank of the root.

Example 3 Referring to the tree in Figure 4.2.1, node $(2, 2, 2, 1)$ is a descendant of node $(1, 3, 1, 1)$, whereas it is a left-relative of node $(3, 1, 2, 2)$. The distance of node $(3, 1, 2, 2)$ from the root is 3, and d^{max} is 3 as well. ○

4.2.2 Pattern and target trees

A pattern is a labeled tree. The pattern is a tree representation of the user interest and can correspond to a collection of navigational expressions on the target tree (e.g., XPath or XQuery expressions in XML documents) or simply to a set of labels for which

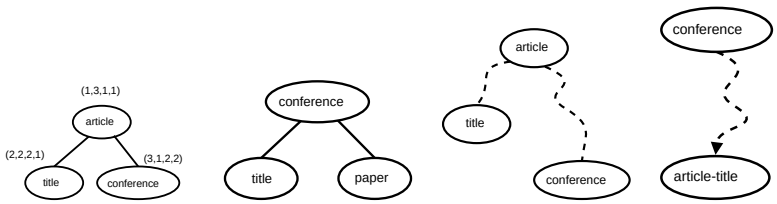


Figure 4.1: (a) Pre/Post order rank, a matching fragment with a different order (b), missing levels (c), and missing elements (d)

a "preference" is specified on the hierarchical or sibling order in which such labels should occur in the target. Labels in the pattern should be semantically distinct each other in order to avoid that two pattern labels can match the same element in the collection.

Example 4 Consider the pattern in Figure 4.2.1. Intuitively, the pattern expresses an interest in document portions related to articles, their titles, and the conferences where they are presented. Possible matches for this pattern are reported in Figure 4.1(b,c,d). The matching tree in Figure 4.2.1 contains similar labels but at different positions, whereas the one in Figure 4.2.1 contains similar labels but at different levels. Finally, the matching tree in Figure 4.2.1 misses an element and the two elements appear at different levels. ○

The target is a set of heterogeneous documents in a source. The target is conveniently represented as a tree with a dummy root labeled `db` and whose subelements are the documents of the source. This representation relies on the common model adopted by native XML databases (e.g., eXist [Meier, 2002], Apache Xindice) and simplifies the adopted notations. An example of target is shown in Figure 4.2. Note that the dummy root has pre-order rank 0 and is at level 0 in the tree.

Definition 3 (*Target*). Let $\{T_1, \ldots, T_n\}$ be a collection of trees, where $T_i = (V_i, E_i)$, $1 \leq i \leq n$. A *target* is a tree $T = (V, E)$ such that:

- $V = \cup_{i=1}^{n} V_i \cup \{r\}$, and $r \notin \cup_{i=1}^{n} V_i$,
- $E = \cup_{i=1}^{n} E_i \cup \{(r, root(T_i)), 1 \leq i \leq n\}$,
- $label(r) =$ `db`.

□

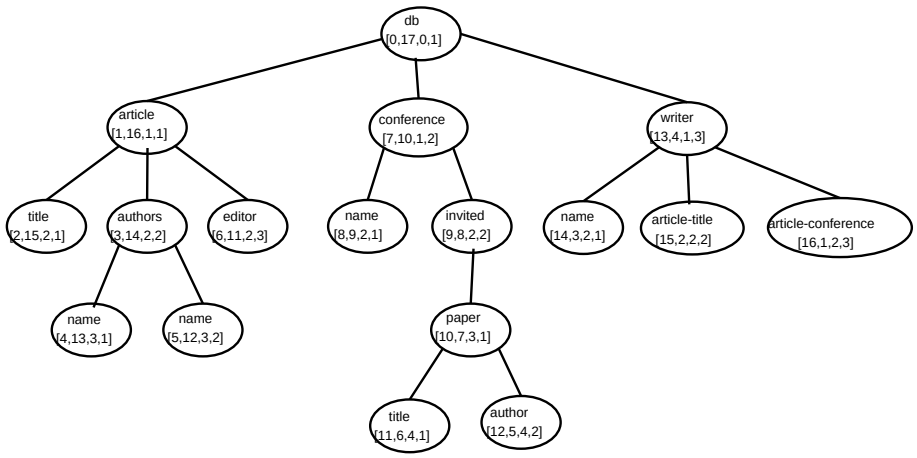


Figure 4.2: A target

4.2.3 Fragment and region trees

The basic building blocks of our approach are *fragments*. Given a pattern P and a target T, a fragment is a subtree of T, belonging to a single document of the target, in which only nodes with labels similar to those in P are considered. Two vertices u, v belong to a fragment for a pattern iff their labels as well as the label of their nearest common ancestor similarly belong to the labels in the pattern. Edges in the fragment either correspond to a direct edge in the target (parent-children relationship) or to a path in the target (ancestor-descendant relationship). Several edges in the target, indeed, can be collapsed in a single edge in the fragment by "skipping" nodes that are not included in the fragment since their labels do not similarly belong to those in the pattern.

Definition 4 (*Fragment*). A *fragment* F of a target $T = (V_T, E_T)$ for a pattern P is a subtree (V_F, E_F) of T for which the following properties hold:

- V_F is the maximal subset of V_T such that $root(T) \notin V_F$ and $\forall u, v \in V_F, label(u), label(v)$, and $label(nca(u, v)) \propto label(\mathcal{V}(P))$;
- for each $v \in V_F$, $nca(root(F), v) = root(F)$;
- $E_F = \{(u, v) \mid u, v \in V_F \wedge (u, v) \in E_T^* \wedge (\nexists w \in V_F, w \neq u, v \text{ s.t. } ((u, w) \in E_T^* \wedge (w, v) \in E_T^*))\}$.

□

Example 5 Consider the pattern in Figure 4.2.1 and the target in Figure 4.2. By considering all the label similarity functions introduced in Section 4.2.6, the corresponding

four fragments are shown in Figure 4.3(a). The first fragment contains nodes whose labels are exactly the same appearing in the pattern. By contrast, the others require to exploit the substring and ontology-based functions. For instance, the second tree contains `paper` as node label, and `paper` $\simeq_{S_o^t}$ `article`. Similarly, the third and fourth trees contain node labels that are similar, exploiting the substring function S_{ss}^t to labels `title` and `conference` in the pattern, respectively. The second tree provides an example of fragment in which a node of the original tree (i.e., node $(9, 8, 2, 2)$ labeled by `invited`) is not included. ○

Starting from fragments, *regions* are introduced as a combination of fragments rooted at the nearest common ancestor in the target. Two fragments can be merged in a region only if they belong to the same document. In other words, the common root of the two fragments is not the `db` node of the target.

Example 6 Consider the tree T rooted at node n $(13, 4, 1, 3)$ in Figure 4.2. It has two subtrees (the ones containing elements `article-title` and `article-conference`) that are fragments with respect to the pattern in Figure 4.2.1. Though n is not (part of) a fragment, the subtree consisting of n and its fragment subtrees could have a higher similarity with the pattern tree in Figure 4.2.1 than its subtrees separately. Therefore, combining fragments into regions may lead to subtrees with higher similarities. ○

A region can be a single fragment or it can be obtained by merging different fragments in a single subtree whose root is the nearest common ancestor of the fragments. Thus, while all fragment node labels similarly belong to labels in the pattern, a region can contain labels not similarly belonging to pattern labels.

Definition 5 (*Regions*). Let $F_P(T)$ be the set of fragments identified between a pattern P and a target T. The corresponding set of regions $R_P(T)$ is inductively defined as follows.

- $F_P(T) \subseteq R_P(T)$;
- For each $F = (V_F, E_F) \in F_P(T)$ and for each $R = (V_R, E_R) \in R_P(T)$ s.t. $label(nca(root(F), root(R))) \neq$ `db`, $S = (V_S, E_S) \in R_P(T)$; where:
  - $root(S) = nca(root(F), root(R))$,
  - $V_S = V_F \cup V_R \cup \{s\}$,
  - $E_S = E_F \cup E_R \cup \{(s, root(F)), (s, root(R))\}$.

□

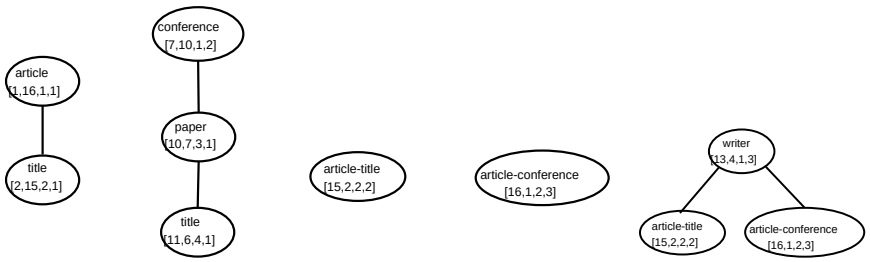


Figure 4.3: (a) Fragments, and (b) generated region

Example 7 The fragments in Figure 4.3(a) are also regions as well as the region reported in Figure 4.3(b) obtained by merging the third and fourth fragments in Figure 4.3(a). Note that the region root label (i.e., `writer`) does not similarly belong to labels in the pattern. ○

This definition of regions results in identifying as regions all possible combinations of fragments in a document of the target. This number can be exponential in the number of fragments. In Section 4.3.3 the *locality principle*, exploited to reduce the number of regions to consider, will be discussed.

The notions of level of a node and distance between two nodes, when applied to regions, refer to the corresponding notions in the original target tree. More specifically, they refer to the target subtree whose root is the region root and whose leaves are the region leaves. We will refer to this tree as the target subtree *covered* by the region. This subtree, as already discussed, may contain additional internal nodes that are not included in the region since their labels do not appear in the pattern. Specifically, internal nodes are included in the covered subtree if they are either in the path from the region root to some region node or if they are internal sibling of two nodes in the covered tree (i.e., right-sibling of one of them and left-sibling of the other).

Definition 6 (*Covered Subtree*). Let T be a target and R be a region on it. The subtree of T *covered* by R, denoted as $C(R)$, is the subtree of T such as:

- $V_{C(R)} \subseteq V_T$ is inductively defined as follows:
  - $V_R \subseteq V_{C(R)}$;
  - $\forall v \in V_T$ such that $\exists u, w \in V_{C(R)}$ and (u, v), $(v, w) \in V_T$, $v \in V_{C(R)}$;
  - $\forall v \in V_T$ such that $\exists u, w, f \in V_{C(R)}$ and $(f, u), (f, v), (f, w) \in V_T$, u is a left-sibling of v, and w is a right-sibling of v, $v \in V_{C(R)}$;
- $E_{C(R)} = \{(u, v) | (u, v) \in E_T s.t. u, v \in V_{C(R)}\}$.

□

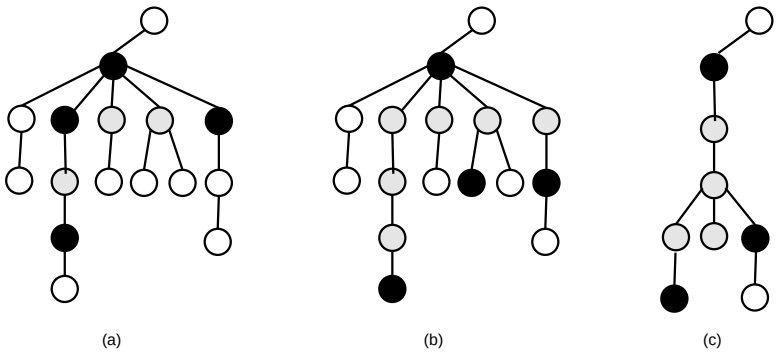


Figure 4.4: Covered subtrees in a target

Example 8 Consider region R_2 of Figure 4.5. The depth of the region $level(R_2) = 4$ though the region includes only three nodes. Similarly, the level of the `paper` labeled node is 3. The distance of the `paper` labeled node is 4, which is also the maximal distance in the region. ○

Figure 4.4 presents different parts of a target where black nodes identify region nodes and gray nodes together with black nodes form the covered subtree.

In this section we present the foundation of our two-phase approach to identify regions similar to a pattern in a target. We first identify the possible matches between the vertices in the pattern and the vertices in the region having similar labels, without exploiting the hierarchical structure of the tree. Then, the hierarchical structure can be taken into account to select, among the possible matches, those that are structurally more similar. Specifically, in this section, after having introduced the definition of mapping, we propose three different similarity measures. Two of them are structural similarity measures, in that they take the tree structure into account to tune the similarity degree obtained by tag matches.

4.2.4 Mapping between a pattern and a region

A mapping between a pattern and a region is a relationship among their elements that takes the tags used in the documents into account. Our definition differs from the definition of mapping proposed by other authors ([Nierman and Jagadish, 2002, Buneman et al., 1997]). Since our focus is on heterogeneous structured data, we do not take the hierarchical organization of the pattern and the region into account in the definition of the mapping. We only require that the element labels are similar.

Definition 7 (*Mapping* M). Let P be a pattern and R be a region subtree of a target T. A mapping M is a partial injective function between the vertices of P and those of R such that $\forall x_p \in \mathcal{V}(P), M(x_p) \neq \perp \Rightarrow label(x_p) \simeq label(M(x_p))$. □

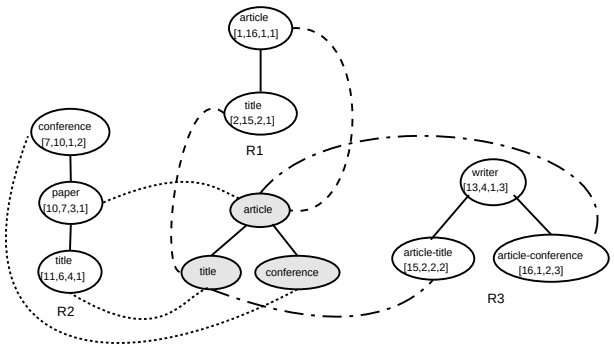


Figure 4.5: Mapping between the pattern and different regions

Example 9 Figure 4.5 reports the pattern P of Figure 4.2 in the center and three target regions, R_1, R_2, R_3. Dashed lines represent a mapping among the vertices of the pattern and those of each region. ○

Several mappings can be established between a pattern and a region. The best one will be selected by means of a similarity measure that evaluates the degree of similarity between the two structures relying on the degree of similarity of their matching vertices.

Example 10 Figure 4.6 reports how different mappings can be established between the region R in Figure 4.3(b) and the pattern P in Figure 4.2.1. ○

Given a similarity measure, like the ones discussed in the next section, assessing the similarity between vertices, a mapping can be evaluated as follows.

Definition 8 (*Evaluation of a Mapping* M). Let M be a mapping between a pattern P and a region R, and let $\mathcal{S}im$ be a vertex similarity function. The evaluation of M is:

$$\mathcal{E}val(M) = \frac{\sum_{x_p \in \mathcal{V}(P) s.t. M(x_p) \neq \perp} \mathcal{S}im(x_p, M(x_p))}{|\mathcal{V}(P)|}$$

□

The similarity between a pattern and a region is then defined as the maximal evaluation among the mappings that can be determined between the pattern and the region.

Definition 9 (*Similarity between a Pattern and a Region*). Let $\mathcal{M}$ be the set of mappings between a pattern P and a region R. The similarity between R and P is defined as:

$$\mathcal{S}im(P, R) = max_{M \in \mathcal{M}} \mathcal{E}val(M)$$

□

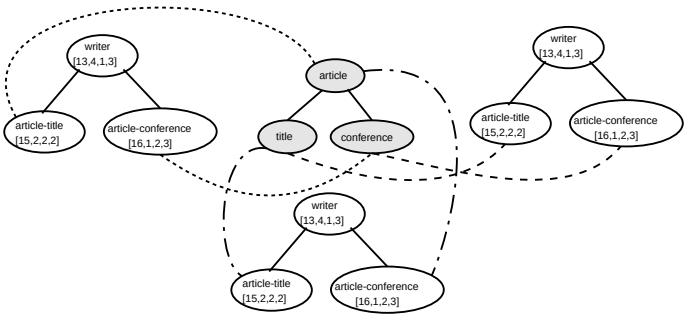


Figure 4.6: Identification of different mappings from the same region and pattern

4.2.5 Similarity between matching vertices

We present three possible approaches for computing the similarity between a pair of matching vertices. The first one assesses similarity only on the basis of label matching, whereas the other two take the structure into account.

4.2.5.1 Match-based similarity

In the first approach, similarity only depends on node labels. Similarity is 1 if labels are identical, whereas a pre-fixed penalty δ is applied if labels are similar. If they are not similar, similarity is 0.

Definition 10 (*Match-based Similarity*). Let P be a pattern, R be a region in a target T, x_p a node of P, and $x_r = M(x_p)$. Their similarity is computed as:

$$\mathcal{S}im_M(x_p, x_r) = \begin{cases} 1 & \text{if } label(x_p) = label(x_r) \\ 1-\delta & \text{if } label(x_p) \simeq label(x_r) \\ 0 & \text{otherwise} \end{cases}$$

□

Example 11 Let x_p be the vertex tagged `article` in the pattern P in Figure 4.5 and x_r^1, x_r^2, x_r^3 the corresponding vertices in the regions R_1, R_2, R_3. Table 4.1(a) reports, in the first column, the match-based similarity between x_p and the corresponding vertices in the three regions.
Table 4.1(b) reports, in the first column, the match-based evaluations of the mappings between P and each of the regions (see Figure 4.5), computed according to Definition 8. For regions R_1 and R_2, there is a single mapping (M_1 and M_2, respectively). For region R_3, by contrast, the three mappings M_3^l, M_3^r, and M_3^b, appearing in the left, right, and bottom, respectively, of Figure 4.6, are evaluated. The table also highlights, in bold, the similarity of P with each of the regions, computed according to Definition 9. ○

	$\mathcal{S}im_M$	$\mathcal{S}im_L$	$\mathcal{S}im_D$
x_r^1	1	1	1
x_r^2	$1-\delta$	$\frac{1}{2}-\delta$	$\frac{1}{2}-\delta$
x_r^3	$1-\delta$	$\frac{1}{2}-\delta$	$\frac{1}{3}-\delta$

	$\mathcal{S}im_M$	$\mathcal{S}im_L$	$\mathcal{S}im_D$
$\mathbf{M_1}$	$\frac{2}{3}$	$\frac{2}{3}$	$\frac{2}{3}$
$\mathbf{M_2}$	$1-\frac{\delta}{3}$	$\frac{7}{12}-\frac{\delta}{3}$	$\frac{1}{2}-\frac{\delta}{3}$
M_3^b	$\frac{2}{3}\cdot(1-\delta)$	$\frac{1}{2}-\frac{2}{3}\cdot\delta$	$\frac{1}{2}-\frac{2}{3}\cdot\delta$
M_3^l	$\frac{2}{3}\cdot(1-\delta)$	$\frac{1}{2}-\frac{2}{3}\cdot\delta$	$\frac{5}{9}-\frac{2}{3}\cdot\delta$
$\mathbf{M_3^r}$	$\frac{2}{3}\cdot(1-\delta)$	$\frac{2}{3}\cdot(1-\delta)$	$\frac{2}{3}\cdot(1-\delta)$

Table 4.2: (a) Similarity of matching vertices (b) Evaluation of mappings and similarity of a pattern with regions

4.2.5.2 Level-based similarity

In the second approach, the match-based similarity is combined with the evaluation of the level at which x_p and $M(x_p)$ appear in the pattern and in the region. Whenever they appear in the same level, their similarity is equal to the similarity computed by the first approach. Otherwise, their similarity linearly decreases as the number of levels of difference increases. We recall that levels in the region refer to the levels in the target subtree *covered* by the region.

Definition 11 (*Level-based similarity*). Let P be a pattern, R be a region in a target T, x_p a node of P, and $x_r = M(x_p)$. Their similarity is computed as:

$$\mathcal{S}im_L(x_p, x_r) = \mathcal{S}im_M(x_p, x_r) - \frac{|level_P(x_p) - level_R(x_r)|}{max(level(P), level(R))}$$

The similarity is 0 if the obtained value is below 0. □

Example 12 The second columns of Tables 4.1(a) and 4.1(b) report the level-based similarities. ○

4.2.5.3 Distance-based similarity

Since two nodes can be in the same level, but not in the same position, a third approach is introduced. The similarity is computed by taking the distance of nodes x_p and $M(x_p)$ with respect to their roots into account. Thus, in this case, the similarity is the highest only when the two nodes are in the same position in the pattern and in the region. We recall that distances in the region refer to the distances in the target subtree *covered* by the region computed through the recursive function d_R in Figure 4.7. Given a region R, $v_1, ..., v_n$ are the vertices of R ordered according to their pre-order rank in the target T. In the figure we report the computation of the distance for the root of R (v_1) and for a generic vertex of R that is not the root. Note that, $d_R^{max} = d_R(v_n)$. Given two

$$d_R(v_1) = 1$$

$$d_R(v_i) = d_R(v_{i-1}) + \begin{cases} level(v_i) - level(v_{i-1}) & v_i \in desc(v_{i-1}) \\ pos(v_i) - pos(v_{i-1}) & v_i \text{ sibling of } v_{i-1} \\ pos(av_i) - pos(av_{i-1}) + level(v_i) - level(av_i) & \text{otherwise} \end{cases}$$

Figure 4.7: Distance of a vertex in a region R

adjacent vertices v_{i-1} and v_i that are not siblings, av_{i-1} and av_i denote their common sibling ancestors. We remark that the last two expressions for computing the distance of a generic v_i in R includes the two previous cases in the definition of d_R. However, for the sake of clarity we have pointed out these two particular cases.

Definition 12 (*Distance-based Similarity*). Let P be a pattern, R be a region in a target T, x_p a node of P, and $x_r = M(x_p)$. Their similarity is computed as:

$$\mathcal{S}im_D(x_p, x_r) = \mathcal{S}im_M(x_p, x_r) - \frac{|d_P(x_p) - d_R(x_r)|}{max(d_P^{max}, d_R^{max})}$$

The similarity is 0 if the obtained value is below 0. □

Example 13 The third columns of Tables 4.1(a) and 4.1(b) report the distance-based similarities. ○

4.2.6 Label similarity

Labels can be similar or dissimilar depending on the adopted criteria of comparison specified by means of functions. In this chapter, the following similarity functions have been considered, even if other ones can be easily integrated:

- *Case insensitive similarity function* S_{ci}^t. Two tags are similar if their differences depend only on the case (e.g., `author` and `Author` are similar).
- *Stemming function* S_{st}^t. Two tags are similar if one is a stem of the other (e.g., `author` and `authors` are similar).
- *Edit distance function* S_{ed}^t. Two tags are similar if their edit distance is less than a prefixed threshold (e.g., `author` and `auth` are similar).
- *Substring function* S_{ss}^t. Two tags are similar if the first one is contained in the second one (e.g., `author` and `conference-author` are similar).
- *Ontology-based function* S_o^t. Two tags are similar if they are synonym relying on a given Thesaurus (e.g., `author` and `writer` are similar).

Relying on these similarity functions, the notions of similarity between two tags or between a tag and a set of tags are defined as follows.

Definition 13 (*Similarity and Similarly Belonging*). Let $\mathcal{S}$ be a set of label similarity functions. Let l_1, l_2 be two labels, l_1 is *similar* to l_2 according to $\mathcal{S}$ (denoted as $l_1 \simeq_{\mathcal{S}} l_2$) if and only if $l_1 = l_2$ or l_1 is similar to l_2 according to a function in $\mathcal{S}$. Let then l be a label and L be a set of labels, l *similarly belongs to* L according to $\mathcal{S}$ (denoted as $l \propto_{\mathcal{S}} L$) if and only if $\exists n \in L$ s.t. $l \simeq_{\mathcal{S}} n$. □

4.3 Construction of fragments and regions

The focus of this section is on the data structures and algorithms for the efficient identification of fragments and regions in the target. As specified in Definition 4, each fragment is a set of nodes bound by the ancestor-descendant relationship in the target. A general purpose indexing structure along with an indexing structure depending on the pattern P are employed for improving the performances of our approach. Fragments are merged into regions, as specified in Definition 5, only when the similarity between P and the generated region is greater than the similarity between P and each single fragment. Thus, regions in the target are single fragments or combinations of regions with fragments. Target subtrees covered by a region should be evaluated only accessing nodes in the regions and information contained in the auxiliary indexing structures.

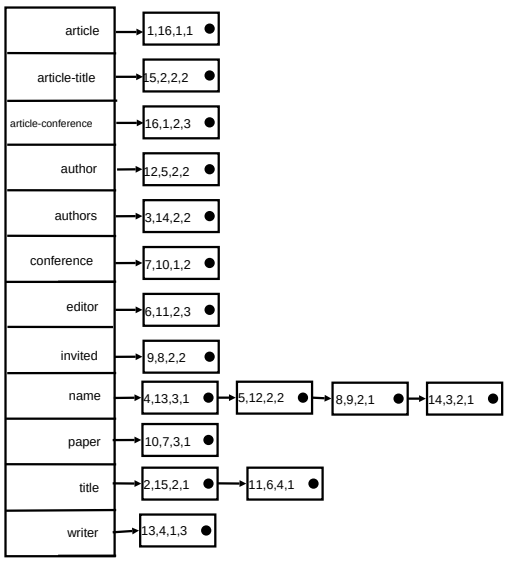
In the remainder of the section, we first present the indexing structures. Then, we discuss the algorithms for the construction of a list of fragments and we present the algorithm for the creation of regions starting from such a list.

4.3.1 Inverted index and pattern index

Directly evaluating the pattern on the target is inefficient and introduce scalability issues in the approach, due to the tag and structural heterogeneity of the collection. For these reasons, an *inverted index* is proposed. This index is independent from the retrieval pattern and is composed by a traditional inverted index coupled with a table, *name-similarity table*, that specifies relationships among tags in the collection relying on the semantic functions discussed in Section 4.2.

This index allows us to easily identify in the collection nodes with similar tags according to different criteria. Since the number of labels that normally occur in a target is sensibly smaller than the number of elements, the size of the name-similarity table is often contained and it can also fit in main memory. This last claim has been proved (check details in Section 4.4) by considering many collections of XML documents gathered from the Web.

The inverted index is built as follows. Starting from the labels of a target, a traditional inverted index is created, that is, each distinct label l occurring in the target is associated



(a)

tag	S^t_{st}	S^t_{ci}	S^t_{ed}	S^t_{ss}	S^t_o
article				1	1
article-title				1	
article-title				2	
article-conference				1	
article-conference				3	
author	1		1		2
authors	1		1		2
conference				3	
editor					
invited					
name					
paper					1
title				2	
writer					2

(b)

Figure 4.8: Semantic inverted index: (a) inverted index and (b) name-similarity table

with the list of vertices labeled by l, ordered according to the pre-order rank. For each vertex $v \in \mathcal{V}(T)$ the 5-tuple $(pre(v), post(v), level(v), pos(v), \mathcal{P}(v))$ is maintained. Then, tags in the collection are grouped according to each of the tag-based similarity functions presented in Section 4.2 and each group is progressively numbered. Each tag is finally associated with the list of the group identifiers it belongs to. Figure 4.8(b) reports this association by means of a table. For example, `author` and `authors` belong to the same class according to function S^t_{ed}, whereas `author`, `authors`, and `writer` belong to the same class according to function S^t_o. In this way, when nodes tagged l should be retrieved in the target, the rows in the name-similarity table corresponding to l are extracted, and then, according to one or more similarity measures, all the similar tags are easily identified. In the case l does not belong to the name-similarity table, one of the tag-based similarity functions can be applied to identify a similar tag in the table.

Given a pattern P, for every node v in P, all the occurrences of nodes u in the target tree such that $label(v) \simeq_{\mathcal{S}} label(u)$ are retrieved from the inverted index according to the functions in $\mathcal{S}$, and organized level by level in a *pattern index*. The pattern index therefore depends on the pattern that should be evaluated on the target. The number of levels in the index depends on the levels in T at which vertices occur with labels similar to those in the pattern. For each level, vertices are ordered according to the pre-order rank.

Depending on the label similarity functions in $\mathcal{S}$, different pattern indexes can be

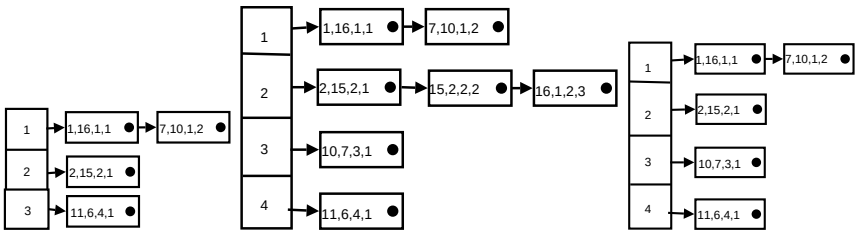


Figure 4.9: Pattern index (a) with label equality, (b) applying ontology-based function, and (c) applying all the criteria

generated. Figure 4.9(a) shows the pattern index obtained by identifying nodes identical to those of the pattern, Figure 4.9(b) reports the pattern index obtained by applying the ontology-based function S_o^t, and, finally, Figure 4.9(c) the pattern index in which all the criteria have been exploited. All the pattern indexes are obtained starting from the pattern in Figure 4.2.1 evaluated on the inverted index in Figure 4.8 corresponding to the target in Figure 4.2.

The following proposition states the number of operations for the construction of a pattern index starting from the inverted index and a pattern.

Proposition 1 Let K be the number of distinct labels in a inverted index SII and M the maximal number of nodes in an entry of SII. The number of operations for the construction of the pattern index for a pattern P is $\mathcal{O}(|P| \cdot K \cdot M)$. △

4.3.2 Algorithms for the construction of fragments

Once the pattern index is generated, the fragments are generated through the recursive Algorithm 1: *CreateFragments* and Algorithm 2: *CreateListOfFragments*. The main advantage of these algorithms is the identification of the fragments through a single visit of the pattern index so that the complexity of fragments construction is kept linear.

To simplify the presentation of the algorithm we assume that, once a node in the pattern index is visited and inserted in a fragment, it is removed from the pattern index. In the algorithms, given a pattern index PI, $size(PI)$ denotes the number of levels, $PI(l)$ denotes the list of nodes at level l ($1 \leq l \leq size(PI)$), and $head(PI(l))$ denotes the first node in the list $PI(l)$.

Algorithm *CreateFragments* is invoked relying on the level and the pre-order rank of the roots of the fragments to be generated. Given a fragment F, *CreateFragments* identifies all the nodes of F appearing in the pattern index and generates a tree on such nodes according to Definition 4. Moreover, the recursive calls of this function also return the fragments whose roots appear in a lower level and precede the root of F.

Relying on the assumption that, when a fragment is generated, it is removed from PI, the PI on which $CreateFragments$ is invoked for a fragment F does not contain all the

Algorithm 1 CreateFragments

Require: $PI, F, v, level$
{PI: pattern index
F: current fragment
v: vertex in current fragment
$level$: a level in the pattern index}
1: $SF = \emptyset$
2: **if** $level \leq size(PI) \wedge head(PI(level))$ is not null **then**
3: **while** $head(PI(level))$ precedes $root(F)$ **do**
4: Create a new fragment $F' = (V'_F, \emptyset)$ such that $V'_F = \{head(PI(level))\}$
5: $SF = SF \cup \{F'\} \cup$ createFragment$(PI, F', head(PI(level)), level + 1)$
6: remove $head(PI(level))$
7: **end while**
8: **while** $head(PI(level))$ descendent of $root(F))$ **do**
9: Insert $head(PI(level))$ in the vertex of F
10: **if** $head(PI(level))$ precedes v **then**
11: Insert $(root(F), head(PI(level)))$ in the edges of F
12: **else**
13: Insert $(v, head(PI(level)))$ in the edges of F
14: **end if**
15: $SF = SF \cup$ createFragment$(PI, F, head(PI(level)), level + 1)$
16: remove $head(PI(level))$
17: **end while**
18: $SF = SF \cup$ createFragment$(PI, F, v, level + 1)$
19: **end if**
Ensure: return SF

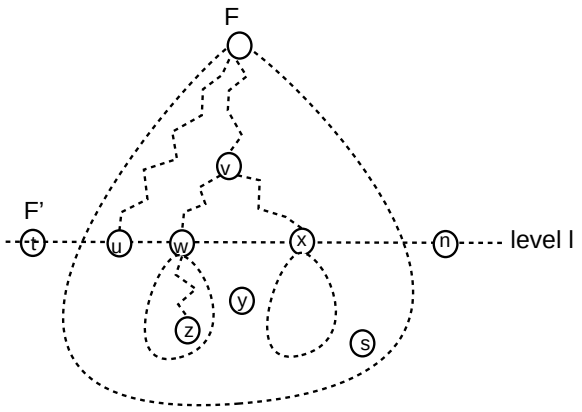


Figure 4.10: Situations that can arise in the creation of a fragment

nodes belonging to fragments whose roots are at a level k such that $k < level(root(F))$. After the invocation of $CreateFragments$ on F, PI will not contain all the nodes of F along with the nodes of the fragments whose roots precede $root(F)$ at a level h such that $(h > l)$.

Algorithm *CreateFragments* takes as input the pattern index PI, the current fragment F, a node v in F (initially the root of F, then an internal node for which we are looking for direct descendants), and the level l in PI where we are looking for descendants of F. Its behavior relies on the following proposition.

Proposition 2 Let $CreateFragments$ be invoked for a fragment F on a level l of a pattern index PI and a leaf node v of F located at a level m in $PI(m < l)$. The following properties hold:

1. If $head(PI(l))$ is a left relative of $root(F)$, $head(PI(l))$ is the root of a new fragment.
2. If $head(PI(l))$ is a descendant of $root(F)$, but not of v, $head(PI(l))$ belongs to F and is a descendant of $root(F)$.
3. If $head(PI(l))$ is a descendant of v in F, $head(PI(l))$ belongs to F and is a descendant of v.
4. If $head(PI(l))$ is a right relative of $root(F)$, no elements of F can be identified at this level.

△

The cases described by Proposition 2 can occur in each invocation of Algorithm $CreateFragments$ and handled as follows. We refer to Figure 4.10 for a better understanding of the behavior of the algorithm. If $head(PI(l)) = t$ is left relative of $root(F)$

(case (1) of Proposition 2), t is the root of another fragment. Indeed, all the fragments F^o such that $root(F^o)$ precedes $root(F)$ at a level k ($k < l$) could contain t have been removed from PI. Therefore, a new fragment F' is created and Algorithm *CreateFragments* is invoked for this fragment and its root at level $l+1$. If $head(PI(l)) = u$ belongs to the descendants of $root(F)$, but it is not a descendant of v (case (2) of Proposition 2), u is left relative of all internal nodes of F. Therefore, u is inserted as child of $root(F)$ and Algorithm *CreateFragments* is invoked on F and u to identify possible descendants of u in F at level $l+1$. If $head(PI(l)) = w$ (the behavior is the same for node x) belongs to the descendants of v (case (3) of Proposition 2), w is associated as a child of v and Algorithm *CreateFragments* is invoked on F and w to identify possible descendants of w in F at level l+1 (as shown in Figure 4.10 node z will be identified). If $head(PI(l)) = n$ is right relative of F (case (4) of Proposition 2), nor n nor the nodes that could follow n at level l can belong to F. Algorithm *CreateFragments* is thus invoked recursively on F, the same v and level $l+1$ in order to identify further descendants of v that do not stay at level l. Through this call, nodes y and s will be identified if v is their immediate ancestor in F.

Starting from the root of a fragment F, Algorithm $CreateFragments$ is recursively invoked on the levels of the pattern index between the level of $root(F)$ and $size(PI)$. Therefore, we are guaranteed that if a level contains a node of F, it is detected and inserted in the correct position in the hierarchical structure of F.

Algorithm $CreateListOfFragments$ generates fragments starting from the first level of PI and moving downwards to its last level. In the first level, all the nodes are roots of fragments. Therefore, Algorithm $CreateFragments$ is invoked on each of the fragments of the first level. Once all the fragments of the first level have been identified, the algorithm proceeds with the nodes of PI still belonging to the remaining levels (if they are not empty). Algorithm $CreateListOfFragments$ ends when it reaches the last level of PI and all the nodes have been removed from PI.

Algorithm 2 CreateListOfFragments

Require: PI
  {PI: pattern index}
1: $SF = \emptyset$
2: **for** $level = 1$ **to** $size(PI)$ **do**
3: **while** $head(PI(level))$ is not null **do**
4: Create a new fragment $F = (V_F, \emptyset)$ such that $V_F = \{head(PI(level))\}$
5: $SF = SF \cup \{F\} \cup$ createFragment$(PI, F, head(PI(level)), level + 1)$
6: remove $head(PI(level))$
7: **end while**
8: **end for**
Ensure: return SF

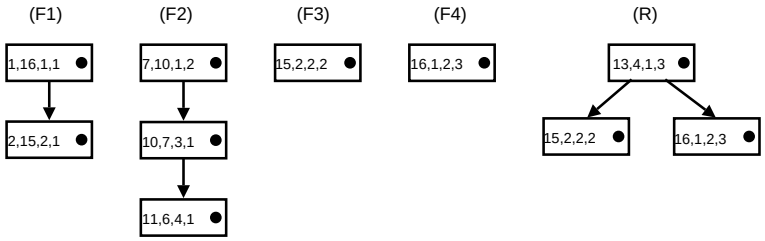


Figure 4.11: Construction of fragments and regions

Proposition 3 Let PI be the pattern index for a pattern P. Algorithm *CreateListOfFragments* correctly identifies all fragments in PI. △

The algorithm visits each vertex in the pattern index only once by removing in each level the vertices already included in a fragment. Its complexity is thus linear in the number of vertices in the pattern index.

Proposition 4 Let PI be a pattern index. The number of operations performed by Algorithm 2: $CreateListOfFragments$ is $\mathcal{O}(N)$, where N is the number of nodes in PI. △

Figure 4.11 illustrates fragments $F_1, \ldots, F_4$ obtained from the pattern index of Figure 4.9(c).

4.3.3 Algorithm for the construction of regions

Two fragments should be merged in a single region when, relying on the adopted similarity function, the similarity of the pattern with the region is higher than the similarity with the individual fragments.

Whenever a document in the target is quite big and the number of fragments is high, the regions that should be checked can grow exponentially. To avoid such a situation we exploit the following *locality principle*: merging fragments together or merging fragments to regions makes sense only when the fragments/regions are close. Indeed, as the size of a region tends to be equal to the size of the document, the similarity decreases.

In order to meet such locality principle regions are obtained by merging adjacent fragments. Operatively, two adjacent fragments can be merged when their common ancestor v is not the root of the target. If it is not the root of the target, their common ancestor v becomes the root of the region and the roots of the fragments become the direct children of v. We remark that the common ancestor of two fragments can be easily obtained by traversing the $\mathcal{P}$ link included in the nodes of the pattern index.

Combining the locality principle and the approach for merging together two adjacent fragments, Algorithm 3: $CreateListOfRegions$ is obtained. The algorithm works on the

list of fragments SF obtained from Algorithm $CreateListOfFragments$ that is ordered according to the pre-order rank of the roots of the fragments it contains. Once a possible region R_i is obtained, by merging two adjacent fragments $SF(i-1)$ and $SF(i)$, the similarity $\mathcal{S}im(P, R_i)$ is compared with the maximal value between $\mathcal{S}im(P, SF(i-1))$ and $\mathcal{S}im(P, SF(i))$. If $\mathcal{S}im(P, R_i)$ is the highest, $SF(i-1)$ is removed from the list and $SF(i)$ substituted with R_i. Otherwise, $SF(i-1)$ is kept alone and we try to merge $SF(i)$ with its right adjacent fragment. The process ends when all the fragments in the list have been checked.

Algorithm 3 CreateListOfRegions

Require: SF, P
$\{SF$: list of fragments
P: Pattern$\}$
1: **for** $i = 2$ **to** $size(SF)$ **do**
2: Try to generate region R_i by merging $SF(i-1)$ and $SF(i)$
3: **if** R_i has been generated **then**
4: **if** $\mathcal{S}im(P, R_i) \geq max\{\mathcal{S}im(P, SF(i-1)), \mathcal{S}im(P, SF(i))\}$ **then**
5: Remove $SF(i-1)$
6: Substitute $SF(i)$ with R_i
7: **end if**
8: **end if**
9: **end for**
Ensure: return SF

Example 14 Considering the running example we try to generate regions starting from the fragments in Figure 4.11. Since the common ancestor between F_1 and F_2 is the root of the target, the two fragments cannot be merged. Same behavior for fragments F_2 and F_3. Since the common ancestor between F_3 and F_4 is a node in the same document, region R in Figure 4.11 is generated. Since the similarity of P with R is higher than its similarity with F_3 and F_4, R is kept and F_3, F_4 removed. At the end of the process we have regions $\{F_1, F_2, R\}$. ○

A key point of our approach is the efficient computation of the similarity between a pattern P and fragment F or a region R. An array indexed on the labels of P is employed to keep the best evaluation of similarity between a node in P and all the nodes in F/R with a similar label. The evaluation in the array are finally added to obtain the similarity between P and F/R. The number of operations is $\mathcal{O}(|P|)$. Since the similarity between P and F can be computed during the construction of a fragment, its complexity in this phase is constant. Things are different for the similarity between P and R, where R is the combination of two fragments. In this case, the evaluations of two arrays should be compared and thus the number of operations is effectively $\mathcal{O}(|P|)$.

Proposition 5 Let SF be the list of fragments extracted for a pattern P from a pattern index PI. The creation of regions from SF requires $\mathcal{O}(N \cdot |P|)$ operations, where N is the number of nodes in PI. △

We wish to remark that the construction of regions is quite fast because the target should not be explicitly accessed. All the required information are contained in the inverted indexes. Moreover, thanks to our *locality principle* the number of regions to check is proportional to the number of fragments. Finally, the regions obtained through our process do not present all the vertices occurring in the target but only those necessary for the computation of similarity. The evaluation of vertices appearing in the region but not in the pattern is computed through the pre/post order rank of each node.

Example 15 Consider the region R_2 in Figure 4.5 and the corresponding representation F_2 in Figure 4.11. Vertex `invited` is not explicitly present in R. However, its lack can be taken into account by considering the levels of vertex `conference` and vertex `paper`. ○

The following proposition summarizes the complexity of the algorithm for the creations of regions starting from a pattern.

Proposition 6 Let P be a pattern, K the number of distinct label in the inverted index SII, and M the maximal size of an entry in SII. Moreover, let N be the number of nodes in the pattern index PI. The number of operations for the creation of regions starting from a pattern is $\mathcal{O}(max\{(|P| \cdot K \cdot M), (N \cdot |P|)\})$. △

4.4 Experimental evaluation

We have used this system to experimentally evaluate, both on real and synthetic data, the following aspects of our approach:

1. Reasonable performance with large collections.
2. Handling of severe structural distortions.
3. Retrieval effectiveness in highly heterogeneous collections.

In the remainder of the section we report our results along these aspects.

4.4.1 Performance evaluation

The performance of the system has been tested using two large synthetic datasets, with associated test patterns. Dataset 1 is designed to contain just a few matching results, embedded in a large number of *don't-care* nodes (around 7500 relevant elements out of

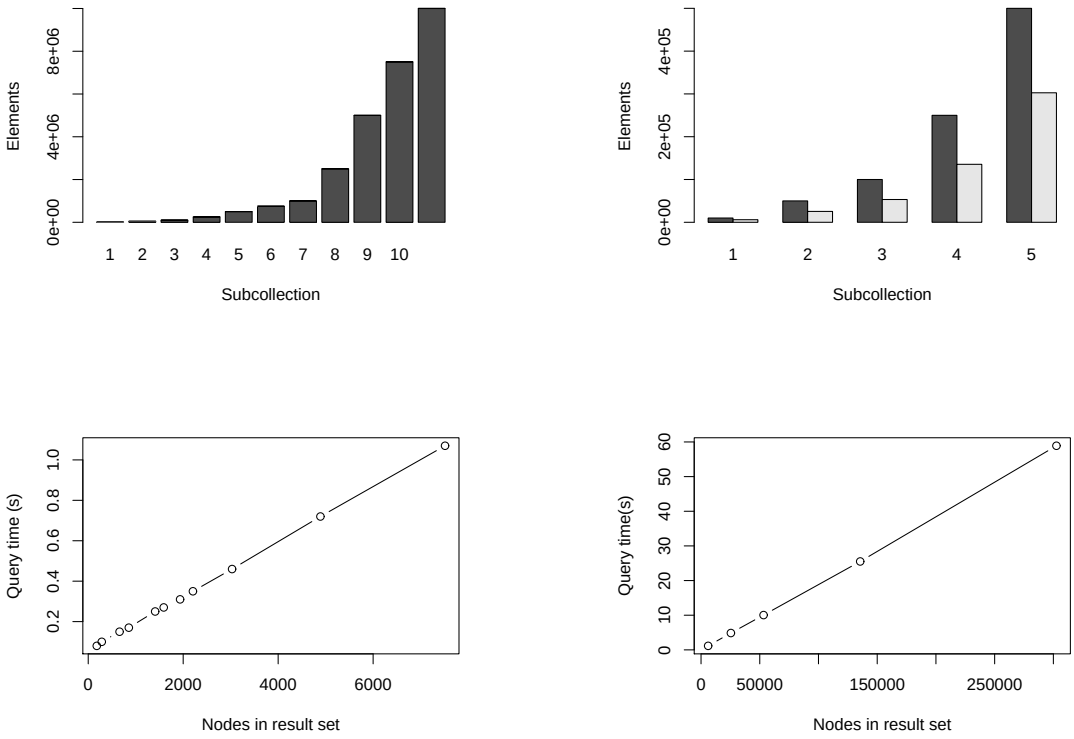


Figure 4.12: (a) Total number of elements in each subcollection extracted from dataset 1 (b) Total number of elements (■) and number of relevant elements (□) in each subcollection extracted from synthetic dataset 2 (c) Execution time in dataset 1 (d) Execution time in dataset 2

10^7 elements). In contrast, dataset 2 has a high proportion of relevant elements (3×10^5 out of 5×10^5). In order to obtain results for a range of dataset sizes, smaller collections have been obtained by sampling each dataset. The characteristics of all datasets are summarized in Figures 4.12(a) and 4.12(b). Results in Fig.s 4.12(c) and 4.12(d) show that the retrieval performance is linearly dependent on the size of the result set.

4.4.2 Effect of structural distortions

The second aspect we have evaluated is the effect of structural variations in fragments. In order to test this, we have generated another synthetic dataset, in which we have embedded potential matches of a test pattern containing 15 elements with the following

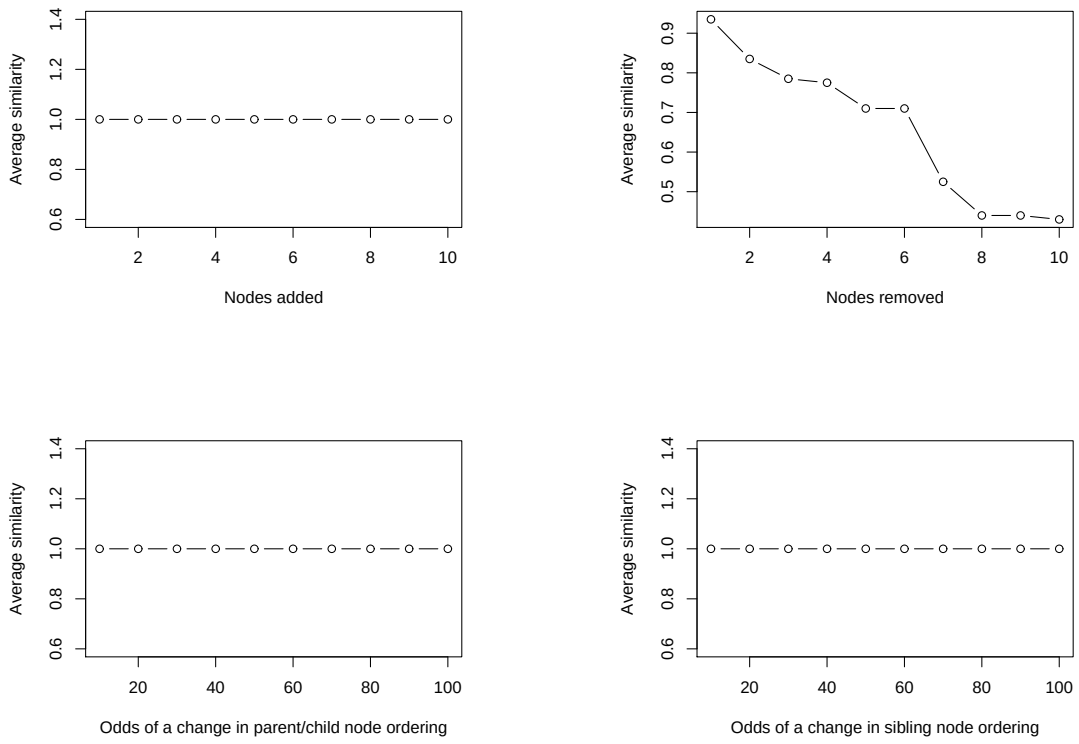


Figure 4.13: Change in similarity with the addition and removal of nodes in regions

kinds of controlled distortions: *(1)* addition of n random nodes; *(2)* deletion of n random nodes; *(3)* switching the order of nodes in the same level, with a given probability; *(4)* switching parent and child nodes, with a given probability.

Results in Fig. 4.13 show that the system is able to find all relevant fragments despite the introduced distortions. Predictably, only the removal of relevant nodes in the target has an effect in the average relevance of results. Adding nodes, switching nodes in the same level, and interchanging parent and child nodes have no effects on the retrieval rate.

4.4.3 Retrieval in highly heterogeneous collections

In our third set of experiments we analyze the retrieval effectiveness in the context of highly heterogeneous collections. Our goal is to evaluate the effectiveness of our approach according to different parameters, such as the size of the data collection, its degree of vocabulary and structural heterogeneity, and its degree of conformance to the query.

Constructor	Pattern Examples	Generated XML
a:genPattern	<a:genPattern a:id='1'> <address/> </a:genPattern>	<address/>
sequence	<t1><a:sequence> <a/><b/> </a:sequence></t1>	<t1> <a/><b/> </t1>
xor	<t1 a:xor='yes'> <a/><b/> </t1>	*a)* <t1><a/></t1> *b)* <t1><b/></t1>
combi	<a:combi> <a/><b/> </a:combi>	*a)* <a_b/> *b)* <a/> *c)* <b/>
if-ancestor	<a:if-ancestor a:name='t1'> <tag2/> </a:if-ancestor>	<t1>... <tag2/>...</t1>
types	<person a:types='user,client'/>	*a)* <user> *b)* <person> *c)* <client>
dmin, dmax	*a)* <t1><t2 a:dmin='0' a:dmax='0'/></t1> *b)* <t1><t2 a:dmin='2' a:dmax='2'/></t1>	*a)* <t1_t2> *b)* <t1_t2> <t1><kjkkij><t2/> </kjkkij></t1> </t1_t2>
a:subPattern	<t1><a:subPattern a:id='1'/></t1>	<t1><address/></t1>

Table 4.3: Constructors for pattern generators

We experimented on a set of document collections patterned after the highly heterogeneous ASSAM[1] dataset. While the information content of the generated collections is the same of that of the real ASSAM, the new collections are bigger and present different levels of heterogeneity and conformance to the queries. The generation is based on empirical estimations of the relative probabilities of ASSAM's main topics and entities. Since the queries present a tree structure as well, pattern generators have been employed for the automatic construction of synthetic queries. The use of a probabilistic approach for the generation of queries simulates the users' uncertainty about the structure of the target collection in query formulation.

To obtain quality measures such as precision and recall we need relevance assessments. These are manually evaluated on the original ASSAM collection. In the generated collections, query identifiers are linked to the pattern used for the generation of the collection, thus specifying which generated subtrees constitute a correct answer for the query. This allows us to build collections of arbitrary size while being able to assess the relevance of each query answer.

In what follows, we first describe the generation of documents and queries and then present the experimental results on the original collection and on the generated ones.

[1] http://moguntia.ucd.ie/repository/datasets/

Generation of documents and queries. To generate suitable test collections and queries we have built a new system whose generation model significantly expands ToXgene [Barbosa et al., 2002] facilities, that only provides limited support for "random structures", for the specification of heterogeneous contents.

Our collection generator provides a set of *XML pattern constructors*, which encapsulate different templates for generating XML structures. A *generator pattern* is itself an XML document, whose nodes represent both pattern constructors and the tags to be generated. In general, the ancestor/descendant relationships between these tags will be preserved in the generated documents. Each node in a generator pattern may have associated a probability indicating the likelihood of finding the sub-tree it represents in the generated documents (if omitted, the probability is taken to be 1.0). Thus, if node u is a child of node v, then $P(u|v) = P(u) \cdot P(v|v')$, where v' is the parent of v. If u is the pattern root node, then $P(u)$ is the probability of the whole pattern. This simple model supports a wide variety of possible heterogeneous structures; Table 4.3 shows the main pattern constructors available.

Table 4.4 shows the summary of the generation patterns used in our experiments. They mainly reflect the features of the ASSAM dataset. As this dataset is mainly a semi-structured representation of a set of web services, their structures are quite heterogeneous and tag names also present many variations. In this collection, tag names are usually phrases that combine in some way the key concepts of the web service. This feature has been simulated using the `a:combi` pattern constructor, estimating n-gram probabilities from ASSAM data. This has been simulated by introducing subpatterns with the constructor `a:subPattern`. In Table 4.4 the last column indicates the used subpatterns in each generator pattern. Two sample generators models are shown in Fig. 4.14.

Results on the generated documents and queries. The generated queries were evaluated against a generated collection of 50000 documents, with a total of one million nodes and around five thousand unique labels (around 500 times larger than the original ASSAM). Given the characteristics of the collection, we selected the following parameters for the experiments:

- We compared the performance of *strict* label matching with a *partial* matching function adapted to the characteristics of the collection as described above. The resulting partial matching function combines most of the similarity functions described in Section 4.2.6.

- We used the structural distance similarity measure described in Section 4.2.5, with $\delta = 0.1$; the results obtained with the level measure were similar for this particular collection.

Pattern	ID	Tags	Depth	Uses
Database Patterns				
Weather Info	BD-1	31	4	Address
Postal Address	BD-2	14	4	-
Stock Quotes	BD-3	16	4	-
Bank Info	BD-4	9	8	Address, Stock
Credit Card	BD-5	8	3	Bank Info
Personal Data	BD-6	19	8	Address, Bank Info
Query Patterns				
Weather	Q-1	12	3	-
ZIP Codes	Q-2	3	1	-
PostBox	Q-3	3	1	-
Quotes	Q-4	5	2	-
Bank Code	Q-5	3	1	-
ATM Location	Q-6	3	2	Address
Contact Person	Q-7	5	2	-
Employee Salary	Q-8	3	2	-
Credit Card	Q-9	3	2	-
User account	Q-10	4	2	-

Table 4.4: Database and query patterns used in experiments

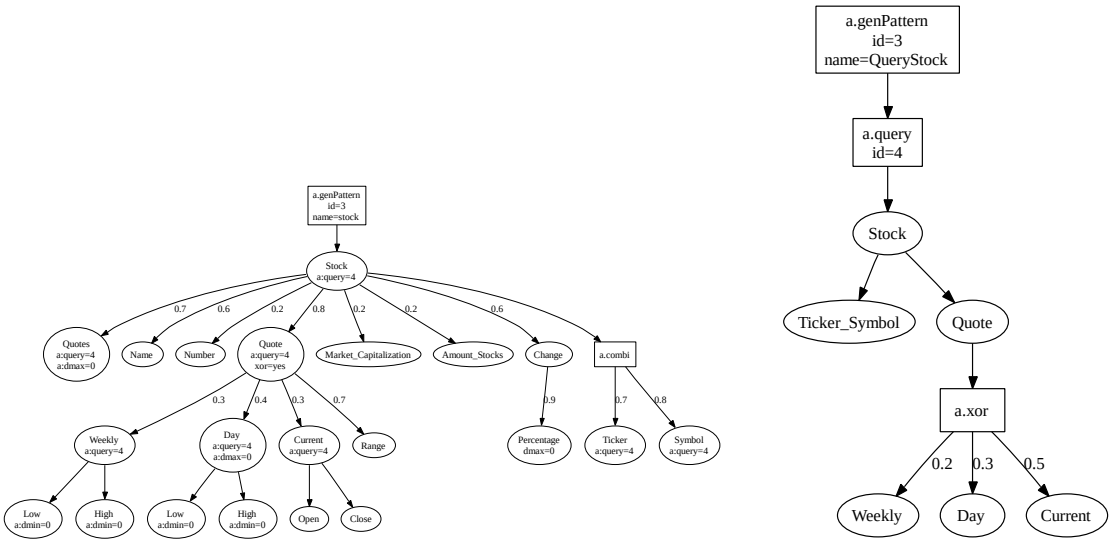


Figure 4.14: (a) Generator for database pattern 3 (b) Generator for query 4, associated with pattern 3

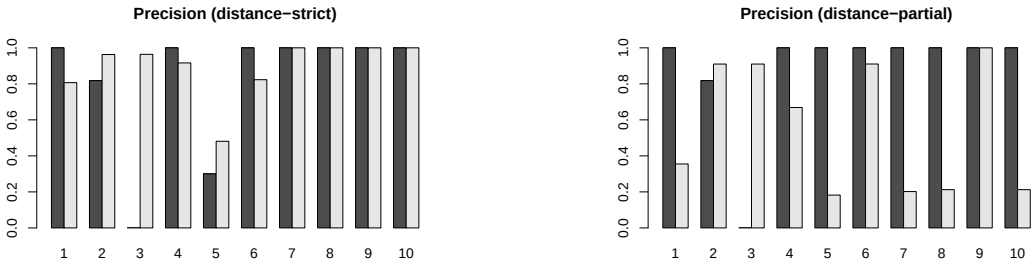
Fig. 4.15 shows the precision, recall and F_1-measure, as well as the precision@10, recall@10 and F_1@10 (i.e. the values express the 10 highest-ranked results). The results under these experimental conditions can be summarized as follows:

- Setting the cutoff at 10 produces much better results than considering all the results. There is an exception for query 3; a closer analysis of this particular case shows that query 3 contains very ambiguous tags, producing a large set of irrelevant results despite the use of tag similarity functions.
- As expected, the results obtained using strict label matching produce a better precision than those obtained using partial label matching, while partial matching produces better recall. Overall, the F_1 measure is generally better for strict matching on all the results, but the combination of 10-highest ranked and partial matching is the best combination.

Results on the original ASSAM collection. Analogous queries were evaluated against the original ASSAM collection, and the relevance of the results checked by hand. A summary of the results is shown in Table 4.16. The first column indicates the query number, and the second column indicates the similarity threshold used for the answer set (we used an explicit cutoff percentage $MinSim$ instead of selecting the k highest results due to the relatively small size of the collection). The results are closely related to those obtained in our more general experimental setting, including the low F_1 for query Q3.

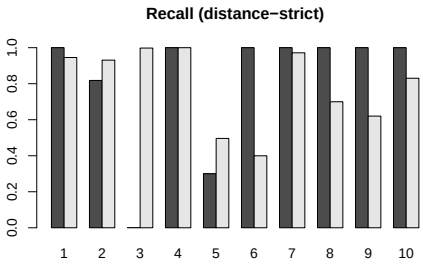
4.5 Concluding remarks

This chapter has developed an approach for the identification of subtrees similar to a given pattern in a collection of highly heterogeneous semi-structured documents. In this context, the hierarchical structure of the pattern cannot be employed for the identification of the target subtrees but only for their ranking. The approach supports the flexible similarity measures described in the previous chapter, and uses specific indexing structures to improve the performance of the retrieval.

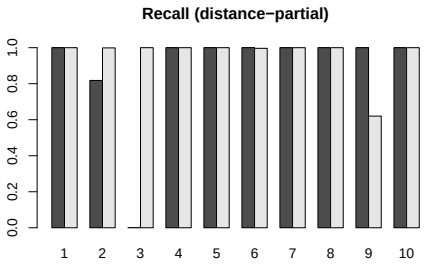


(a)

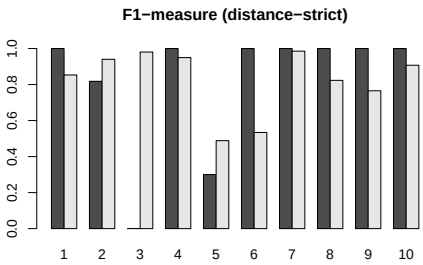
(b)



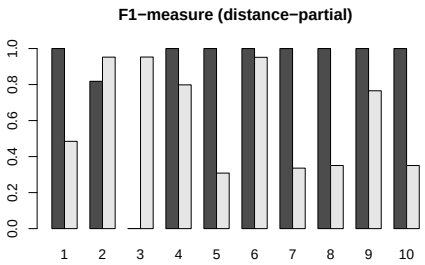
(c)



(d)



(e)



(f)

Figure 4.15: Precision, recall and F_1 for measure distance, using strict and partial label matching. For each of the 10 queries, the darker bars (■) are computed using only the highest-ranked 10 results, the lighter bars (□) consider all the returned matches.

Query	$MinSim$	$Prec.$	$Recall$	F_1
Q1 (Weather)	0.2	1	0.75	0.86
Q3 (Addresses)	0.15	0.29	0.7	0.41
Q4 (Bank data)	0.2	1	0.8	0.89

Figure 4.16: Results for the ASSAM datasets

Chapter 5

Extensions

The algorithms presented in the previous chapter serve as a foundation for approximate querying of XML with flexible similarity measures. This chapter presents contributions that provide refinements which are important in practice. Section 5.1 discusses the integration of fragment-based queries into XQuery, and provides a function-based specification for the incorporation of the new techniques presented in this thesis into existing XQuery engines. Then, Section 5.2 presents a top-k version of the algorithms which is experimentally shown to provide an important performance boost without loss of generality. In contrast to existing algorithms, this approach does not require the similarity measures to be monotone. Finally, Section 5.3 presents an incremental, representative-based clustering algorithm suited to organize collections of heterogeneous fragments into coherent groups.

5.1 Integration into XQuery

To be useful in practice, the algorithms presented in chapter 4 must be incorporated into an XML processing framework. This means that they should be part of the repertoire of an XQuery processing technique. In addition, XQuery can be extremely useful as a tool for post-processing the fragments and regions retrieved using approximate techniques.

Example 16 Consider a fragment search over a database consisting of multiple documents. A data analyst may wish to view the results grouped by document, returning only the most relevant fragment in each document and sorting by the average score of all matches in each document. These data aggregation operations can be easily expressed using XQuery conditions over the resulting fragments.

○

5.1.1 Requirements

This section presents the design requirements for an extension of XQuery that can support fragment-based approximate structural querying:

1. The changes should *not* require major syntactic changes to be implemented into an existing XQuery engine. There is a consensus that XQuery parsers are complex enough already.

2. The changes should be a proper extension of XQuery; that is, they should not change the standard "exact" query behavior. That is, approximate query semantics must be *explicitly* requested.

3. It should be possible to specify which similarity measure to use on a case-per-case basis, or even to combine several distinct similarity measures in the same expression.

4. XQuery compositionality must be preserved: that is, all standard operations that are possible on XPath expressions should also be possible on retrieved fragments.

5. It should be possible to retrieve fragment-specific information about the retrieved fragments (the *docId*, and so on).

6. Useful operations for fragments and regions, such as finding the nearest common ancestor of two fragments, should be available.

5.1.2 Functional vs. keyword approach

There are essentially two approaches to address the requirements proposed in the previous section: either to extend the language with new keywords or to specify a library of functions. The main advantage of introducing new keywords is that it clearly points out that the underlying semantics is changed, together with the optimizer. In contrast, using functions may indicate "externally defined" operations without effective access to the optimizer. Of course, this is not necessarily the case, since the XQuery specification defines several fundamental primitives using functions; in particular, the simplified XQuery Core sublanguage, into which XQuery implementations typically translate queries for optimization, uses functions rather that keywords extensively[Fankhauser, 2001].

As a consequence, and especially with regard to the first requirement outlined above, we define a library of functions. Syntactic sugar can always be added afterwards.

5.1.3 Specification

To be able to define the necessary functions, we need to define first the appropriate types. These are:

Namespace declarations We will assume that all definitions are enclosed in the ahx namespace, declared as follows:

```
declare namespace ahx = "http://krono.act.uji.es/arhex/ns";
```

Type definitions In order to represent types and fragments in XQuery we will assume that the following types are defined:

- `ahx:measure` represents a similarity measure. Note that it is possible to serialize instances of our measure model as XML, which opens the possibility of introspection: defining and managing measures using XQuery itself.
- `ahx:fragment` represents a fragment. For all purposes it should be considered equivalent to `xs:anyType`, since in general fragments do no conform to an schema.

Function definitions Figure 5.1 lists the XQuery signatures of a a set of functions that correspond to the requirements outlined above. Their informal semantics is as follows:

- `ahx:load_measure` obtains an instance of `ahx:measure` from an external source.
- `ahx:fragments` returns a list of fragments that correspond to a pattern using a given measure.

Example 17 The following query performs the approximate query:

```
let $pattern := <writer>
<article-title/><article-conference/>
</writer>
let $measure := ahx:load_measure('distance.measure')
return ahx:fragments($pattern, $measure)
```

○

The following functions access information about the retrieved fragment:

- `ahx:score` returns the score of a fragment.
- `ahx:docid` returns the document id of the fragment.
- `ahx:docname` returns the document name of the fragment.

Example 18 The following query returns fragments with a score greater than 0.5, ordered by score

```
let $pattern := <writer>
<article-title/><article-conference/>
</writer>
let $measure := ahx:load_measure('distance.measure')
for $fragment in ahx:fragments($pattern, $measure)
where ahx:score($fragment) > 0.5
order by ahx:score($fragment) descending
return $fragment
```

○

```
declare function ahx:load_measure($source as xs:string) as ahx:measure external;
declare function ahx:fragments($pattern as xs:string, $measure as ahx:measure) external;
declare function ahx:score($fragment as ahx:fragment) as xs:float external;
declare function ahx:all_nodes($fragment as ahx:fragment) as ahx:fragment external;
declare function ahx:subtree($fragment as ahx:fragment) as ahx:fragment external;
declare function ahx:docid($fragment as ahx:fragment) as xs:integer external;
declare function ahx:docname($fragment as ahx:fragment) as xs:string external;
declare function ahx:nca($fragments as ahx:fragment+) as ahx:fragment external;
```

Figure 5.1: XQuery Declarations of the proposed fragment-oriented functions

Example 19 The following query returns the list of document names that are present in the query results.

```
let $pattern := <writer>
<article-title/><article-conference/>
</writer>
let $measure := ahx:load_measure('distance.measure')
for $d in distinct-values(for $x in ahx:fragments($pattern, $measure) return ahx:docid($x))
return ahx:docname($d)
```

○

Finally, the following functions provide useful operations on fragments:

- `ahx:all_nodes` returns a copy of a fragment that includes all *don't-care* nodes.
- `ahx:subtree` returns a copy of a fragment that includes all nodes under the fragment root.
- `ahx:nca` returns the nearest common ancestor of a list of fragments

Example 20 The following query returns the minimal complete subtree that contains all fragments in the result, for each document occurring in the results of the approximate query.

```
let $pattern := <writer>
<article-title/><article-conference/>
</writer>
let $measure := ahx:load_measure('distance.measure')
let $fragments = ahx:fragments($pattern, $measure)
for $d in distinct-values(for $x in $fragments return ahx:docid($x))
return ahx:subtree(ahx:nca(for $f in $fragments where ahx:docid($f) = $d return $f))
```

○

5.1.4 Discussion

A proof-of-concept implementation of the previous functions have been produced, and it has been incorporated into the prototype presented in Chapter 6. The prototype uses the IBM XQuery Normalizer and Static Analyzer[1] for parsing and translation into (an early version of) XQuery Core. A simple non-optimizing query engine processor is used to execute the queries on top of B^+-tree indexes.

This prototype demonstrates the feasibility of including approximate query facilities in a standard XQuery implementation at a lexical level. The incorporation of an approximate pattern operator in a standard algebraic optimization framework is an open problem, but some research has been done in the incorporation of *exact* pattern-oriented operators in practical algebras. In particular, [Michiels et al., 2007] proposes extending the Galax [Fernández et al., 2003] algebra with a TupleTreePattern operator, which abstracts the processing of a complete twig pattern, instead of dividing it up into a series of steps. In principle, it should be possible to encapsulate the region-oriented algorithms presented in the previous chapter using an approximate version of the TupleTreePattern operator.

Our experimentation with the prototype in practical settings has shown that most useful queries include some grouping. XQuery does not include an explicit *group-by* operator, and grouping is done using a standard idiom which is illustrated in the previous examples. This makes it difficult to exploit group-by based optimizations. In this respect, the incorporation of an approximate variant of the approaches presented by [Beyer et al., 2005] and [Gokhale et al., 2007] could provide performance benefits.

In addition, approximate queries may involve measures that return highly heterogeneous result sets, thus making it difficult to apply standard grouping techniques, In these cases, more sophisticated techniques such as clustering may be necessary to postprocess the results. An algorithm suited for its incorporation in this context is described in Section 5.3.

5.2 Top-k processing

Using similarity functions naturally leads itself to top-k processing, which allows the retrieval of only the best results. In this section we address the problem of efficiently evaluating top-k fragment pattern queries with ad-hoc ranking functions. For instance, consider the following query:

```
let $fragment := <writer>
<article-title/><article-conference/>
</writer>
let $measure := ahx:load_measure('distance.measure')
for $fragment at $rank in ahx:fragments($fragment, $measure):
where $rank <= 10
```

[1] http://www.alphaworks.ibm.com/tech/xqnsta

```
return $fragment
```

A brute-force strategy for solving this query is to generate all results, sort them by score and return the k results with the highest score. A slight refinement, presented in algorithm 4, saves space by using a heap to keep only the best k results found so far. We will use this algorithm as a baseline to compare against more elaborate implementations.

Algorithm 4 Naïve top-k

Require: p: pattern, m: measure, k: integer
  $heap \leftarrow \emptyset$ {empty heap with capacity k}
  for each $f = generateNextResult(pattern)$ **do**
    $fragment.score = m(p, f)$
    $heap.insert(f)$ {use score as value}
  end for
  return All fragments in the *heap*

The general problem of top-k ranging is based on the concept of finding an *aggregate* score based on an aggregation of scores at a lower granularity level; in a relational context, this usually means computing the total score based on the value of several attributes in a table. The best-known algorithm for top-k processing in general is Fagin's *threshold algorithm [Fagin et al., 2003]*, also called a *sort-merge* approach since it is based on scanning the lists of pre-sorted attributes and merging them into a total score until in can be guaranteed that the top k objects have been found. Unfortunately, this approach requires the aggregation function to be monotonic, which is too restrictive for our componentized measure framework. As an alternative, [Xin et al., 2007] propose a progressive *index-merge* approach (also in the relational context), in which a progressive search is performed over a space of joint states composed by multiple attributes. This removes the requirement for monotonicity.

Example 21 Many useful variations of the simple sum aggregation function are non-monotonic. Consider a tie component C with an associated similarity function f_C, which takes as arguments two other components $C_1$and C_2 at a lower granularity level, with associated similarity functions $f_{C1}$and f_{C2} respectively. In this case, the following similarity functions are all non-monotonic:

- A function that subtracts the score of the second component function: $f_C(o_1, o_2) = \max\{f_{C1}(o_1,o_2) - f_{C2}(o_1, o_2), 0\}$.
- A function that penalizes the deviation of a similarity score from 0.5: $f_C(o_1, o_2) = \max\{f_{C1}(o_1,o_2) - |f_{C2}(o_1, o_2) - 0.5|, 0\}$.

○

Note that searching over the solution space will potentially generate all solutions; in the worst case, this is equivalent to the naïve algorithm. This approach can be improved by cutting off the search with appropriate criteria which guarantee that, once a certain state is reached, further searches will never yield results in the top-k. This means that we need to be able to know, at least, what is the *maximum* possible value that can be achieved by the similarity function, given a *partial* instantiation of a pattern.

5.2.1 Problem statement

In our case, the space of solutions is the space of *fragments and their mappings* to the target, as defined in Definition 7. As a consequence, the search process needs to construct fragments progressively, together with their relevant mappings.

Algorithm 5 Index-merge top-k

```
Require: p: pattern
  topkHeap ← ∅
  sHeap ← findRoots()
  while sHeap ≠ ∅ do
    S ← top entry from sHeap
    if S is a leaf state then
      Build the solution and update topkHeap
    else
      Find the next children of the node
      Generate the new states and insert them into sHeap
    end if
  end while
  return the top k solutions in topkHeap
```

Algorithm 5 is based on the index-merge approach. It works by progressively matching nodes in the pattern to nodes in the target in a way that maximizes the value of the similarity function. It maintains two heaps: one for the complete top-k mappings found so far (which we name the *states* of the search) and another one for the candidate mappings which are still being build. For each state, the algorithm evaluates the maximum value that can be achieved, given the available information. Those states that cannot reach the current minimum threshold to enter the top-k are pruned, while the rest will be expanded. Once the state cannon be further expanded it is called a *leaf state*, and contains a compete result (i.e. mapping). The state heap guarantees that the most promising candidate mapping (the one with the highest possible final score) will be visited first. Note that the top-k algorithm is designed to search the database for promising candidates in a (roughly) breadth-first way, analogous to the order in which the pattern index is visited by *CreateFragment* algorithm.

In order to adapt the general index-merge algorithm to our XML context, the following questions must be answered:

1. Which are the initial nodes for the computation?
2. How to generate new states?
3. How to create a function that returns the highest possible final score?

The following subsections address each of these points.

5.2.2 Finding roots

Since the solution we seek is a set of fragments and mappings, the obvious candidates for initial states are the set of root nodes. In order to characterize this set, consider the structure of the Pattern Index discussed in Section 4.3. The pattern index is organized by level, and the *CreateFragment* algorithm exploits this fact to guarantee that the fragment roots are always identified before all the successors. More precisely:

Definition 14 (*roots of a pattern index*) Let PI be a pattern index. The roots of the pattern index PI, denoted by $roots(PI)$, are the nodes $n \in PI$ such that there is not any other node $n' \in roots(PI)$ that is an ancestor of n. □

From this definition and the Definition 4 of fragment, the next result follows immediately:

Proposition 7 Let PI be a pattern index. Let LF be the list of fragments created by algorithm 1 using PI as input, and let $roots(LF)$ be the set of roots of all fragments in LF. Then:

$$roots(LF) = roots(PI)$$

△

As a consequence, it suffices to find the set of roots of the pattern index in order to find the possible roots of all solutions. Obviously, it is not practical to actually build the pattern index in order to extract the roots. Instead, we can exploit the fact that different roots are, by definition, in different subtrees, to find a way to progressively construct patterns without actually building the pattern index. This is expressed in the following proposition:

Proposition 8 Let PI be a pattern index and $r_1, r_2 \in roots(PI)$. Then,

$$pre(r_1) < post(r_2) \rightarrow pre(r_2) > post(r_1)$$

△

This results allows us to "jump" in the index to the next candidate root without having to iterate through all the nodes in the subtree of the previous candidate root. We define the algorithm defines an iterator [Garcia-Molina et al., 2000]. An iterator is an abstraction of operations over indexes in which the results are returned one at a time to the caller (using the **yield** statement) instead of as a complete set, which is often impractical in database implementations as it requires materialization[2]. The caller access a iterator using two methods: the *next*() method retrieves the following value, and the *skip*(k) method jumps to the object whose key is at least k (this assumes that the iterator returns ordered values).

The following algorithm, *nextCandidate*, takes as inn put an iterator over nodes in a subtree, ordered by *pre*, and uses the *next* and *skip* methods to define a new iterator that returns the roots in the sense of Proposition 8.

Algorithm 6 nextCandidate

```
Require: nodeIterator
  while nodeIterator not empty do
    node ← nodeIterator.next()
    yield node
    nodeIterator.skip(node.post + 1)
  end while
```

Algorithm *nextCandidate* is the key step in determining the set of all candidate roots, which will determine the initial states. The complete procedure is outlined in Algorithm 7.

Algorithm 7 findRoots

```
Require: iterators
  heap ←empty heap
  for it ∈ iterators do
    heap.insert(it.next())
  end for
  while heap is not empty do
    node ← heap.pop()
    yield node
    val ← it.nextCandidate()
    if if val exists  then
      heap.push(val)
    end if
  end while
```

[2] A well-known application of the iterator technique is the *cursor*.

The algorithm itself takes as an input a set *iterators*, whose elements iterate over the list of vertices labeled by each label l in the pattern, ordered according to the pre-order rank. The heap is used to keep the roots in *pre* order.

Proposition 9 The set of nodes returned by Algorithm 7 is the same as $root(PI)$ for a pattern index PI built using the same set of iterators. △

Sketch Proof of Proposition 1 The results follows from the observation that Algorithm *findRoots* traverses the tree in the same order as algorithm *CreateFragments*. ◇

5.2.3 State generation

We can refine the definition of partially-built mapping by noting that, in our progressive search-based framework, the still unmapped nodes in a a partially built mapping may be put in two categories: either an unmapped node in the fragment *may* have a successor in the pattern that has not been reached yet, or we can guarantee that such a node does not exist. We will denote each of these possibilities by using the special nodes, denoted as $V_?$ and $V_\perp$respectively.

Definition 15 (*Annotated partially built mapping*) An *annotated partially built mapping* is a mapping that allows in its range the special nodes $V_?$ and $V_\perp$. More precisely, let P be a pattern with vertex set $\mathcal{V}(P)$, and R be a region subtree of a target T with vertex set $\mathcal{V}(R)$. An *annotated partially built mapping* is a partial injective function between $\mathcal{V}(P)$ and $\mathcal{V}(R) \cup \{V_\perp, V_?\}$, with the same properties as the M mappings specified in Definition 7. □

Example 22 Figure 5.2 shows an annotated partially built mapping that includes a node c that may be still found in the target, and a node e that definitely does not occur. ○

This allows us to define the *states* that will encapsulate information about the partially built mappings as we progressively search the database.

Definition 16 (*Search state*) Given a target $T =< V_T, E_T >$, a search state S is a tuple $S =< P, m, root(S) >$, where:

1. $P =< V_P, E_P, root(S) >$ is a pattern
2. $root(S) \in V_T$
3. m is mapping function defined as follows:

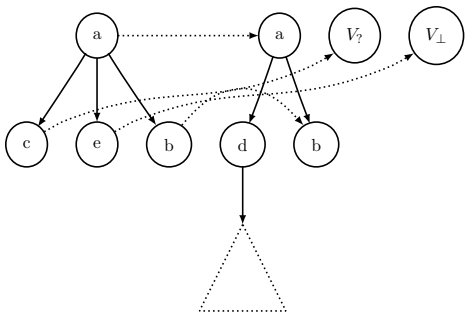


Figure 5.2: An annotated partially built mapping from a pattern (left) into a target (right)

- $m : V_P \rightarrow 2^{V_T \cup \{V_\perp, V_?\}}$; that is, it maps the set of vertices of the pattern to as set of vertices in the target, including the special nodes $V_\perp$ and $V_?$
- $\forall v \in V_P, m(v) = \{v_1, \ldots, v_k | \forall v_i label(v) \simeq label(v_i) \wedge \forall v_i : \bar{root}(S)$ is an ancestor of $v_i\}$

4. $m(v) = V_\perp$if there is no node in V_T that can be mapped to node v in V_P.
5. $m(v) = V_?$ if there no node in V_T has been yet found to map to node v in the pattern, but it is possible that such a node exists.

□

Given the previous definitions, we can define both the initial states and a procedure to obtain new states progressively. The initial states are just the states that contain just a root node obtained using Algorithm 7. Then, Algorithm 8, can be used to obtains the nodes that can be attached to an existing state to derive a new one, by doing a search of the *immediate* successors of the leaf nodes in an existing state.

The pruning of states is driven by the transformation of $V_?$ nodes into $V_\perp$nodes. This is performed by the application of Algorithm *nextCandidate*, which can determine when no more children exist.

5.2.4 State scoring

We can assign a score to a state S using a slightly modified version of a similarity evaluation function for fragments:

Definition 17 (*Maximum evaluation f_{max} of a partially-built mapping*) Let $f : M \rightarrow [0, 1]$ be an arbitrary evaluation function on mappings. A function $f_{max} : M \rightarrow [0, 1]$, is a *maximum evaluation* function for a partially-built mapping if, for each mapping involving $V_?$ and $V_\perp$, it returns the *maximum* potential value of the similarity that can

Algorithm 8 nextCandidateChildren

Require: *state*
  $leafLabels \leftarrow$labels found in the leaf nodes of *state*
  $children \leftarrow \emptyset$
  $iterators \leftarrow \emptyset$
  for each *label* in $leafLabels$ **do**
    $iterators[label] \leftarrow$iterator for *label* positioned at the first successor of the corresponding leaf node in the *state*
  end for
  for each first *node* in *iterators* **do**
    $noChildren \leftarrow true$
    $val \leftarrow i.next()$
    while *val* exists **do**
      $children[node].append(val)$
      $val \leftarrow val.nextCandidate()$
    end while
    if $noChildren$ **then**
      $children[node] \leftarrow V_{\perp}$
    end if
  end for
  return *children*

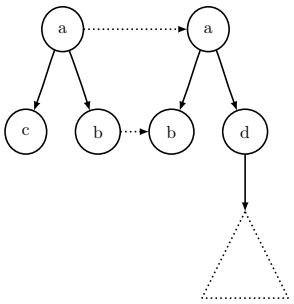


Figure 5.3: A partially built mapping from a pattern (left) into a target (right), which has not been still visited completely

be attained by f once $V_?$ nodes are replaced by actual nodes in the target and taking into account that nodes mapped to $V_\perp$ will certainly not be found.

Maximum evaluation functions will usually be based on "maximum" versions of similarity functions. □

Example 23 The following is the "maximum" version of the match similarity function 10:

$$Max\mathcal{S}im_M(x_p, x_r) = \begin{cases} 1 & \text{if } label(x_p){=}label(x_r) \text{ or } x_r = V_? \\ 1{-}\delta & \text{if } label(x_p){\simeq}label(x_r) \\ 0 & \text{otherwise} \end{cases}$$

○

Example 24 Consider the partially-built mapping shown in Figure 5.3, and assume that nothing is known about the portion of the target that still has not been visited, (except that it is not empty). If the evaluation function f is based on the *match* similarity function, then $f_{max}(pattern, fragment) = 1$, because the missing node may still be found, which would make the value of f equal to 1. In contrast, with the level-based similarity measure, $f_{max}(pattern, fragment) < 1$, as there will be at least a difference of one level between the position of the missing node in the target and the node.

The availability of a function that finds an upper bound to the maximum possible value of the evaluation of a partially-build mapping allows us to cut the computation once it is known that completing a mapping will never attain a similarity level big enough to make it into the top-k fragments. ○

5.2.5 Putting it all together

Algorithm 9 combines all of the results presented in the previous sections.

First, $findRoots$ is used to obtain the initial states. Then, for any given state, some nodes will be mapped and some will not. The following cases are possible:

Algorithm 9 Fragment-based top-k

Require: $index$, k, $fragment$, $f_{partial}$, f
  $topkHeap \leftarrow$empty heap
  $sHeap \leftarrow$empty heap
  $iterators \leftarrow getIterators(fragment)$
  $labelIds \leftarrow getLabelIds(fragment)$
  for each $node$ in $findRoots(iterators)$ **do**
    $state \leftarrow$ new state based on the nodes in $fragment$ with root $node$
    $sHeap.append(state)$
  end for
  while $sHeap$ is not empty **do**
    $state \leftarrow sHeap.pop()$
    $leaves \leftarrow state.getLeaves()$
    $children \leftarrow nextCandidateChildren(mapping)$
    $hasChildren \leftarrow False$
    for each $parentNode$ in $children$ **do**
      for each child ch of $parentNode$ **do**
        $hasChildren \leftarrow True$
        $state.add(ch)$
      end for
    end for
    if $hasChildren$ **then**
      $sHeap.append(state)$
    else
      $topkHeap.append(state)$
    end if
  end while
  return fragments corresponding to the top k states in $topkHeap$

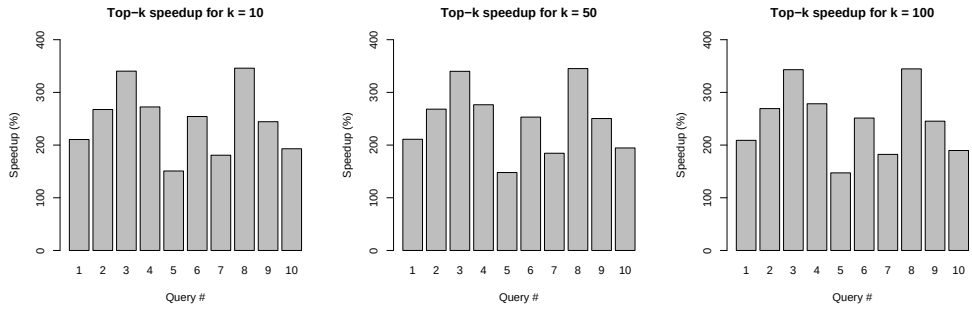


Figure 5.4: Performance improvement using the top-k algorithm with distance-based similarity

- For each node with an assigned match
  - No other matching node is present in the children set. In this case, nothing happens.
  - One or more matching nodes is present in the children set. In this case, the state is duplicated and the new state is added to the state heap.
- For each node without an assigned match
  - No other matching node is present in the children set. In this case, nothing happens.
  - One matching node is present in the children set. In this case, the state is updated with the new match.
  - More than one matching nodes is present in the children set. In this case, new states are created as necessary and added to the state heap.

The computation stops when no more states can be generated whose score is potentially larger than that of the top-k leaf states found so far.

5.2.6 Experimental evaluation

In order to compare the performance of the top-k algorithm with respect to the exhaustive approach, we used exactly the same experimental settings as in Section 4.4, changing only the query processing algorithm. Figure 5.4 shows performance improvements between around 150% and 350%, independently of the size of k (for $k = 10, 50$ and 100).

5.2.7 Discussion

The experimental results show that the top-k algorithm provides a significant performance improvement over the exhaustive version presented in the previous chapter. In addition, since it is based on an index-merge algorithmic framework, it supports non-monotonic similarity functions, and therefore can accommodate flexible, complex measures such as those allowed by the multi-similarity framework presented in Chapter 3.

5.3 Clustering

The algorithms presented in the preceding section and the previous chapter may return, when used with a loose similarity measure, results of high structural complexity. This motivates the need for grouping the results, already mentioned in section 5.1.4. For cases in which the grouping mechanisms including within XQuery are not enough, some form of clustering of the results is usually required.

In this section we propose a clustering algorithm suited to the following requirements:

- It should have direct support for flexible XML similarity measures.
- It should be incremental, in order to be used on line as part of the presentation of query results.
- Able to produce a description of the obtained classes which is meaningful for end users.

Many clustering algorithms that can be adapted to this scenario exist; some good surveys are [Jain et al., 1999, Xu and Wunsch, 2005]. The most stringent requirement is the last one, which forces us to choose a representative-based approach; that is, an algorithm that describes the resulting classes using members of the classes themselves; in our case, document fragments which can be presented to the user. In contrast, a centroid or other abstract representation of a class is difficult to define in the context of XML documents, although some approaches have studied the midpoint of a set of document schemas [Abelló et al., 2005]

To describe the clustering routine, the following definitions are needed:

Definition 18 (*document class*) Each document class C is a set of documents whose structural similarities with respect to the class representatives are greater than a given threshold β_{sim}. The class to which a document d belongs is denoted with $class(d)$. □

Definition 19 (*class representative*) The class representatives, denoted $repSet(C)$, are themselves documents of the class C. The similarity between two representatives of the class must not be greater than a given threshold $\beta_{rep}(\beta_{rep} \geq \beta_{sim})$. □

In this way, each class will represent a set of common structural properties, which are established by the intersection of all its representatives, as well as a set of structural particularities, which are stated by its representatives. Consequently, the threshold β_{sim} determines the degree of optionality of the associated class schema, whereas the number of representatives determines the degree of heterogeneity of the class schema. This is captured by the following proposition, which follows immediately from the definitions above:

Proposition 10 Let $\{d_1 \ldots, d_n, \}$ a set of XML documents, m a similarity measure and C, C' document classes. The following invariants must hold:

1. $\forall d_i \in C, \exists d_r \in repSet(C) : m(d_i, d_r) \geq \beta_{sim}$
2. $\forall d_i \in C, \forall C' : C' \neq C, \forall {d'}_r \in repSet(C') : m(d_i, d'_r) < \beta_{sim}$
3. $\forall d_r, d'_r \in repSet(C) : d_r \neq d'_r \implies m(d_r, d'_r) \leq \beta_{rep}$

△

5.3.1 Algorithm

The proposed clustering routine (Algorithm 10) uses an inverted file over the representatives of the classes in order to efficiently calculate the structural similarity of each incoming document. Each entry of the inverted file represents a path element, and its associated value is the list of representatives having that path element.

The algorithm updates the inverted file and the set of current document classes according to the structure of each new incoming document. It takes into consideration the following situations:

- When the similarity between the new document and several representatives of different classes is greater than the given threshold β_{sim}, all the involved classes must be joined into one single class. The variable New is used for this purpose, which incrementally aggregates the involved classes (lines 3– 16). Additionally, a representative set must be revised for the resulted class (lines 11–15 and lines 21–23).
- When the similarity between the new document and all the current representatives is not greater than the given threshold β_{sim}, the document belongs to a new class, whose representative is the new document itself.
- The representative set of a class must be updated in the following two cases: when the new document structurally subsumes some of its current representatives, and when the similarity between the new document and all the class representatives is not greater than the threshold β_{rep}. In the first case, the new document replaces all

Algorithm 10 Clustering routine

Require: d_{new}, β_{sim}, β_{rep}, m, $InvertedFile$, $Classes$
{d_{new}: new incoming document;
β_{sim}: similarity threshold for documents;
β_{rep}: similarity threshold for representatives;
m: similarity function;
$InvertedFile$: inverted file for document class representatives;
$Classes$: set of currently detected classes;}

1: Select from the $InvertedFile$ the representatives whose similarity with d_{new} is greater than β_{sim}, and put them into the set $Docs$
2: $New \leftarrow \varnothing$
3: **for all** $d \in Docs$ **do**
4: **if** $New = \varnothing$ **then**
5: $New \leftarrow class(d) \cup \{d_{new}\}$
6: $New.repSet \leftarrow repSet(class(d))$
7: **else**
8: $New \leftarrow New \cup class(d)$
9: $New.repSet \leftarrow New.repSet \cup repSet(class(d))$
10: **end if**
11: **if** $\exists d' \in repSet(New)$ such that d_{new} subsumes d' **then**
12: Remove all the representatives of New subsumed by d_{new}
13: Add $d_{new}$to the set of representatives of New
14: **end if**
15: Remove the class $class(d)$ from $Classes$, and all its representatives from the $InvertedFile$.
16: **end for**
17: **if** $New = \varnothing$ **then**
18: Add to $Classes$ the new class $\{d_{new}\}$
19: Update the inverted file with d_{nenew}
20: **else**
21: **if** $\nexists d' \in repSet(new)$such that $docSim(d', d_{new}) > \beta_{rep}$ **then**
22: Add d_{new} to the list of representatives of New
23: **end if**
24: Add to $Classes$ the updated class New
25: Add the representative set of New to $InvertedFile$
26: **end if**

the representatives that are subsumed by it. In the second case, the new document becomes a new representative of the class.

The following propositions establish the correctness and complexity of the clustering algorithm.

Proposition 11 (*Correctness*) Algorithm 10 maintains the invariants described in Proposition 10. △

Sketch Proof of Proposition 2 Each of the invariants can be independently shown to be kept by the algorithm:

1. The set *Docs* is initialized with the representatives whose similarity with the new document is greater than β_{sim}. It the set is empty, then the new document does not affect this invariant. If it is not empty, the classes are fused, which generates a new class set which keeps the invariant.

2. The construction of the New class in lines 1–15 set guarantees that each new document is added to a class in a way that keeps this invariant.

3. This invariant is guaranteed by the condition in lines 20–22.

◇

Proposition 12 (*Complexity*) Let $D = \{d_1, \ldots, d_n\}$ a set of XML documents. Algorithm 10 has a worst case complexity of $O(|D|^2)$ △

Sketch Proof of Proposition 3 The clustering routine is executed $|D|$ times, once for each document in D. The algorithm needs to check all representatives; in the worst case, the number of representatives is also $|D|$, hence the quadratic complexity. ◇

5.3.2 Experimental evaluation

In order to check the effectiveness, of the approach, the algorithm has been tested using the highly heterogeneous ASSAM collection described in Section 4.4.3, which consists of 116 documents. Figure 5.5 demonstrates the progressive creation of new clusters (22 in total) for parameters $\beta_{sim} = \beta_{rep} = 0.4$, using a generic path-based document similarity measure [Sanz et al., 2003].

Since the complexity of the algorithm effectively depends on the number of representatives, a desirable property of the algorithm is that this number is kept as low as possible. Figure 5.6 shows that, except for extreme values of $\beta_{sim}$and β_{rep},the number of representatives stays remarkably constant.

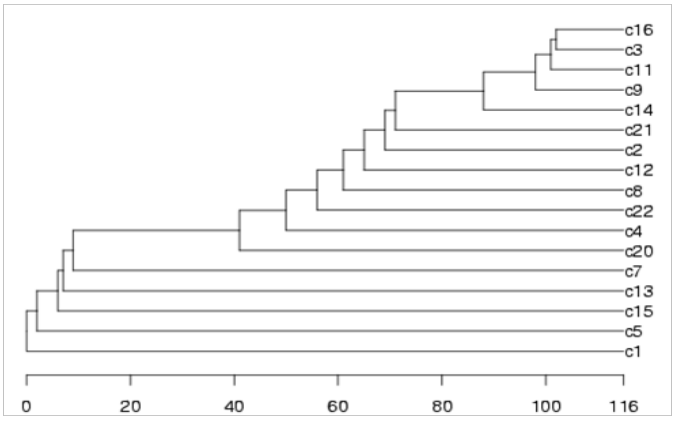


Figure 5.5: Progressive creation of clusters, using $\beta_{sim} = \beta_{rep} = 0.4$

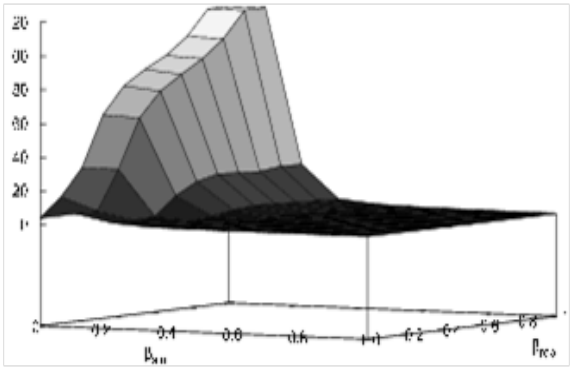


Figure 5.6: Number of representatives

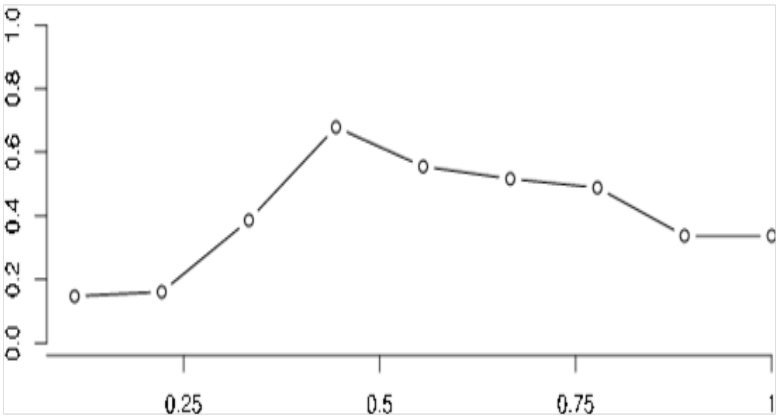


Figure 5.7: Quality of clusters (F_1measure) for a range of values of β_{sim}

Finally, we evaluated the quality of the resulting clusters using the F_1-measure, taking as reference a hand-made classification of all documents in the collection. Figure 5.7 shows the results for a range of values of β_{sim} . The results are consistently good for values of $\beta_{sim} > 0.3$, with a peak in quality at about 0.4. This indicates that, for a given collection, it may be necessary to perform a preliminary study to determine appropriate values of β_{sim} and β_{rep}, much like many partitional clustering algorithms require the *a priori* determination of its parameters, usually the number of desired classes.

5.3.3 Discussion

We have described a clustering algorithm that is appropriate for on-line clustering of complex XML result sets, and therefore is it useful as a component in the multi-similarity querying of collections. The experimental evaluation shows good results for a number of desirable properties. The algorithm is unsupervised, except for the initial determination of two similarity parameters. We do not think, however, that this is an important limitation, since the intended use of this algorithm is as part of interactive query tools, and the parameters can be simply adjusted by immediate user feedback.

5.4 Concluding remarks

The techniques developed in this chapter are designed to improve the applicability of the basic techniques presented in the previous chapter.

Integration into XQuery is achieved using a function-based specification that demands minimal changes on the lexical front-end of an existing XQuery-capable system. A proof-of-concept implementation on a non-optimizing backend has been produced; integration

with the optimizer is a more challenging research question, but recent advances in pattern-oriented optimization should be directly applicable to solve the problem.

The approximate techniques naturally lead to a top-k version of the algorithms. To the best of our knowledge, the techniques presented in this chapter include the first such algorithms for XML that can support non-monotone similarity measures. While there exists a lot of room for optimization (for instance, much more aggressive pruning can be applied if it is known a priori that the similarity measure is in fact monotone), the baseline provides a significant performance boost over the "exact" version of the algorithms.

Finally, a clustering algorithm is presented that can be used for grouping together the results of especially complex queries. Throughout this chapter, the importance of grouping for XML data in contexts where heterogeneity is significant has been pointed out. This has been often neglected, as exemplified by the fact that XQuery does not even include a grouping operation, and relies on an idiomatic approaches that makes optimization of grouping very difficult. As the complexity of available XML data keeps increasing, this is a problem that will be growing in importance, and only recently have researchers begun to tackle it.

Chapter 6

Prototype

The techniques presented in the previous chapters have been implemented as part of *ArHeX*, an experimental toolkit for flexible XML information processing. The prototype is organized as a set of tools, designed to help a data engineer deal with highly heterogeneous collections.

As a case study, we present its application in the context of a biomedical research project, in which it is useful for the exploration of a large collection of biomedical resources with respect to some disease model. Section 6.1 briefly explain the common ArHeX back-end, and the present the two main user-level components of the framework: the *measure editor* (Section 6.2.1) and the *query tool* (Section 6.2.2). Finally, section 6.2 discusses the application of these tools in the context of a biomedical research project.

6.1 ArHeX tools

6.1.1 Architecture and Back-end

Figure 6.1 presents a high-level view of the ArHeX architecture. Its components are organized in three levels: The *user interface level* provides graphical tools for the end user. These tools are based on the components of the *query level, which contains* the core components which implement the algorithms presented in the preceding chapters. At the lowest level, the storage manager provides basic facilities. Each of these levels contains several components; the most important of these are presented next.

User Interface level The User Interface level contains two end-user applications:

- The *Measure Editor* allows the creation and editing of tailored similarity measures, using the available components.
- The *Query tool* is an interface to the diverse query APIs available, allowing users to search for fragments in an XML collection either directly or via the XQuery engine

In addition to these GUI applications, several APIs for embedding the ArHeX capabilities in other applications are also available.

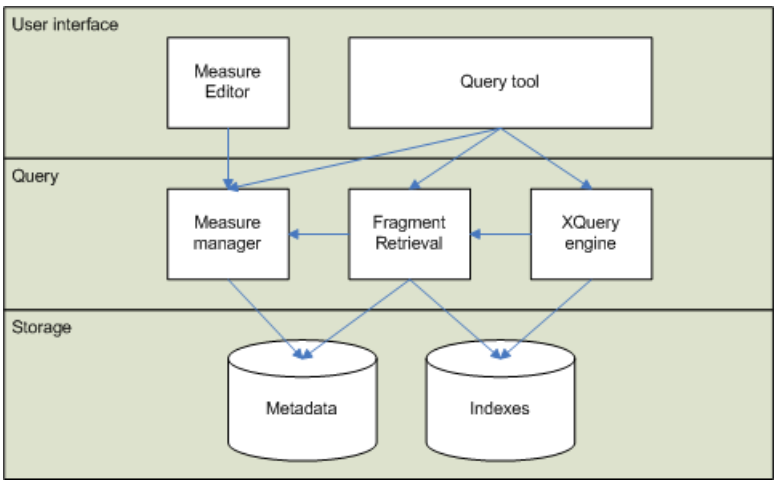


Figure 6.1: ArHeX Architecture

Query level The query level contains the core functionality of the toolkit; in particular, it implements the measure management and the query algorithms presented in this thesis. The main components are:

- The *Measure Manager* controls the available measures. It provides information about the available components and measures in the system, and makes sure that any new measure is indeed valid.
- The *Fragment Retrieval* component implements query algorithms on XML fragments
- The *XQuery engine* provides an XQuery interface over the fragment retrieval, and implements a subset of the standard XQuery facilities such as traditional XPath matching.

Storage level At the lowest level, the system needs to provide support for storing the XML collections and the associated metadata. This requires a series of generic facilities, such as B-trees, and concurrency control. We have used the open source Oracle Berkeley DB, which provides "a transactional storage engine for un-typed data in basic key/value data structures".

6.2 Scenario: Application to a biomedical research project

To show the application of the ArHeX toolkit in a realistic setting, we will present a scenario based on a real biomedical research project, Health-e-Child (IST 027749) [Freund et al., 2006]. Its aims are:

> The Health-e-Child project aims at developing an integrated health care platform for European paediatrics, providing seamless integration of traditional and emerging sources of biomedical information. The long-term goal of the project is to provide uninhibited access to universal biomedical knowledge repositories for personalized and preventive health care, large-scale information-based biomedical research and training, and informed policy making.

More specifically:

- To gain a comprehensive view of a child's health by vertically integrating biomedical data, information, and knowledge, that spans the entire spectrum from genetic to clinical to epidemiological;
- To develop a biomedical information platform, supported by sophisticated and robust search, optimization, and matching techniques for heterogeneous information, empowered by the Grid;
- To build enabling tools and services on top of the Health-e-Child platform, that will lead to innovative and better health care solutions in Europe:
  - Integrated disease models exploiting all available information levels;
  - Database-guided biomedical decision support systems provisioning novel clinical practices and personalized health care for children;
  - Large-scale, cross-modality, and longitudinal information fusion and data mining for biomedical knowledge discovery.

Health-e-Child is a large project, so we will focus on a particular area: the creation of *disease models*, which require a complex integration of biomedical data, information, and knowledge, which integrates information from multiple abstraction layers. For instance, the description of a disease at the *individual, tissu*e and *molecular* levels is completely different; but in an integrated model, the correspondences that can be found between phenomena at all these levels is crucial. For each particular disease model that must be created, one of the required tasks is to find out relevant portions of publicly-available biomedical information repositories which are relevant for the given disease, and should be eventually merged or aligned in the final model. The result is an Integrated Disease Knowledge Model (IDKM), which specifies the concepts of particular diseases, taking into account all the biomedical abstraction layers. Patient-centric information collected in the hospitals will be semantically annotated in terms of a particular IDKM. Figure 6.2 depicts the information contained in a set of four large publicly available biomedical ontologies, showing the relevant abstraction layers.

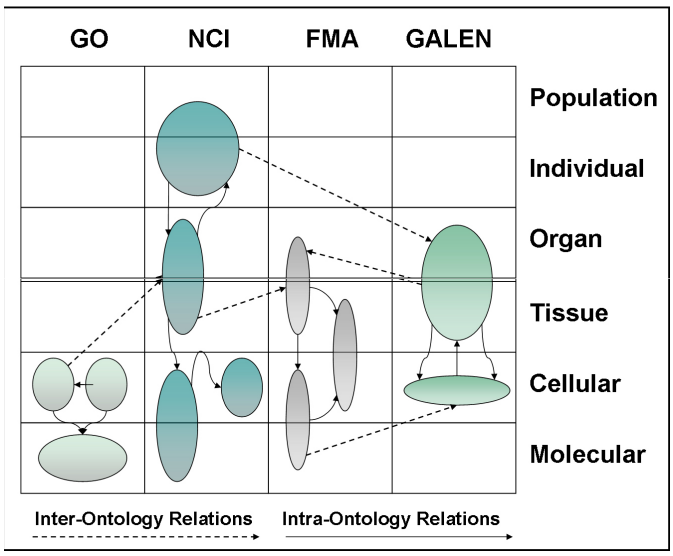


Figure 6.2: Granularity levels in a disease model

This search of relevant sources within large independent sources is not trivial. Firstly, many information repositories in Biomedicine do not cover the requirements of specific applications completely. Moreover, the concepts contained can involve different abstraction levels (e.g. molecular, organ, disease, etc.) that can be in the same or in different domain ontologies. Secondly, independent sources are normally rather large, resulting in two main effects: users find them hard to use for annotating and querying information sources and only a subset of those are used by system applications. Finally, in current integration approaches, it is necessary to manually map the existing data sources to domain concepts, which implies a bottleneck in large distributed scenarios.

As an example, we will show how to use the ArHeX toolkit to retrieve ontology fragments in order to guide the building of the IDKMs. Thus, starting from a collection of ontologies and a knowledge pattern, users can query the knowledge and progressively construct the required IDKM. Figure 6.3 shows how this task fits in the overall methodology [Jiménez et al., 2006]. Section 6.2.1 shows the usage of the Measure Editor, and Section 6.2.2 presents the use of the Query Tool in order to query the aforementioned repositories.

6.2.1 Measure Editor

The queries require appropriate similarity measures, which are built using the *measure editor*, which supports the creation and tailoring of flexible similarity measures that conform to the model presented in Chapter 3. Figure 6.4 shows the main screen of the measure editor. The left pane presents a list of the already-defined measures, while

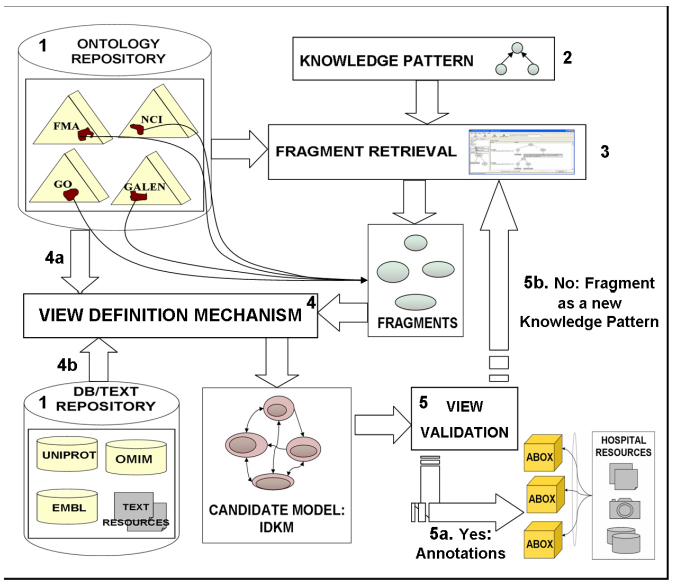


Figure 6.3: Creation of IDKMs in HeC

the right pane presents the top-level metadata annotations that describe the selected measure. In particular, the selected "root component" indicates that the actual measure is an instance of the *RegionEvaluator* component, which operates at the region similarity level. Note also that the measure requires the underlying index to support strict label matching.

Each component can be individually tailored. Figure 6.5 shows the options available for tailoring a particular component. Note that the user interface forms presented to the user are dynamically generated according to the metadata of each component.

The tool supports the graphical creation of complex components. Figure 6.6 shows a "tie" component that implements a weighted sum at an arbitrary granularity level (in this case, at the node level). Note the tree structure of the measure, as displayed in the left pane.

It is indeed possible to define new measures from scratch. In this case, the tool guides the process by exploiting metadata to ensure that the new measure is correctly defined. Figure 6.7 shows how the tool indicates which components are possible, according to the provides/requires hierarchy, at a given point.

6.2.2 Query tool

The query tool is based on an Oracle Berkeley DB-based back-end. It is being used for the exploration of heterogeneous sources in biomedical information integration projects. An

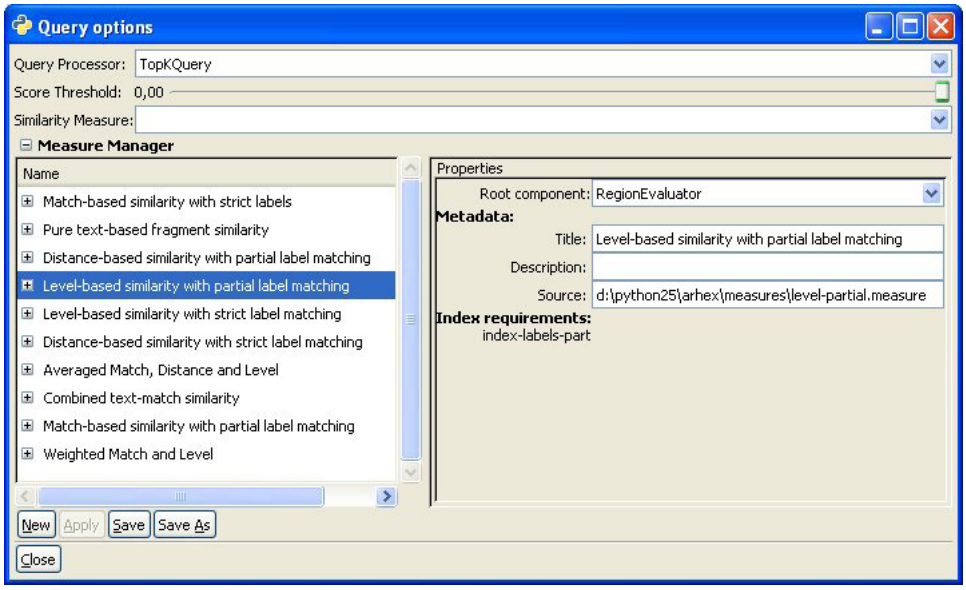


Figure 6.4: Main screen of the measure editor

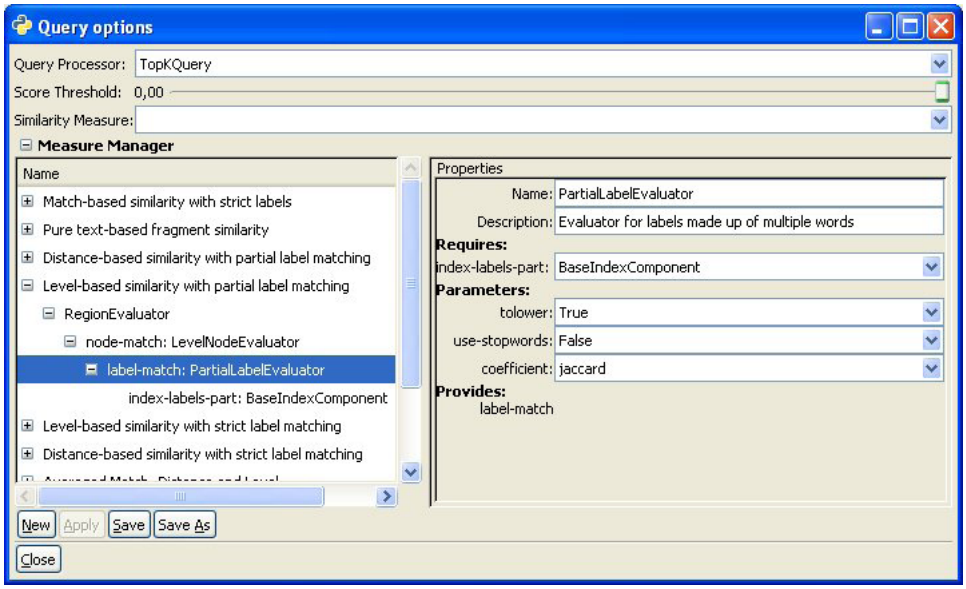


Figure 6.5: Component tailoring

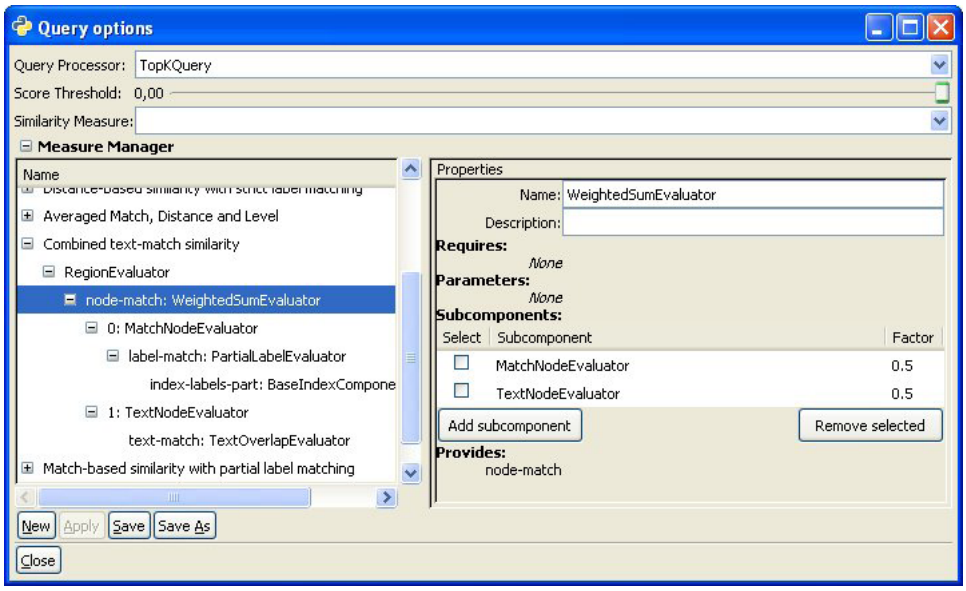


Figure 6.6: Editing a "tie" component

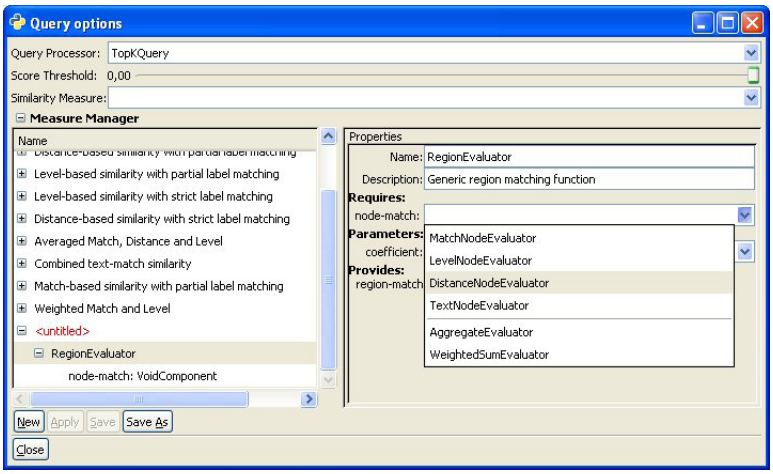


Figure 6.7: Creating a new measure from scratch

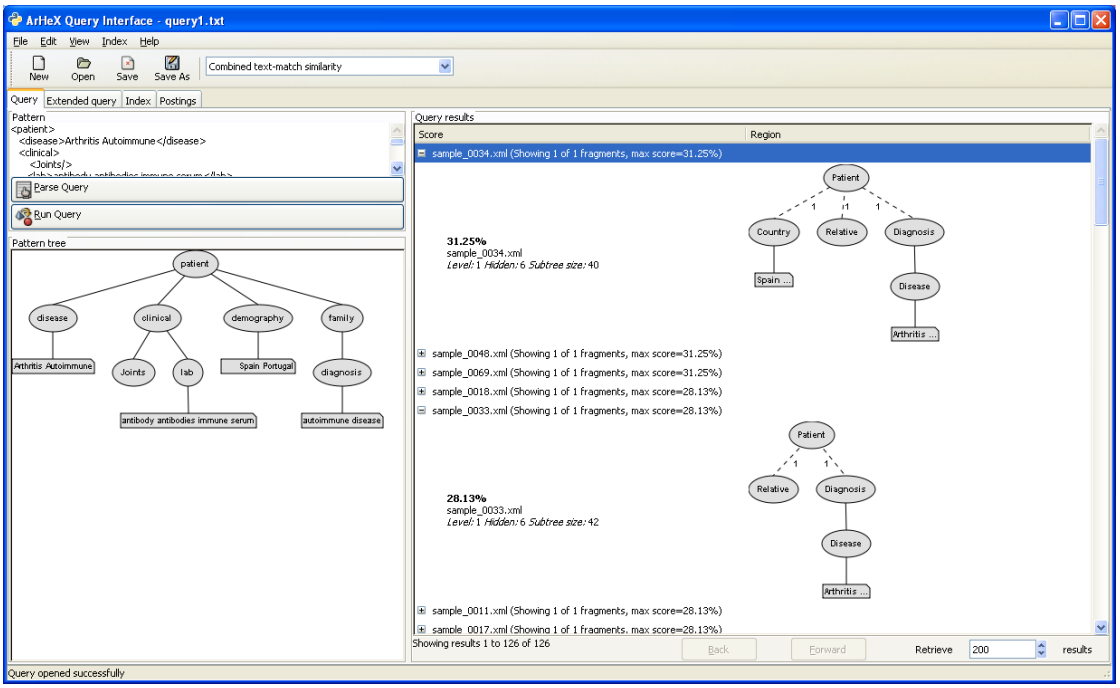


Figure 6.8: Screenshot of the GUI prototype showing the top results in some collections, using a variation of the match-based similarity and a flexible label-matching function

example of this functionality is shown in two screenshots: Figure 6.8 shows the combined results for a query and Figure 6.9 shows the top results found in different collections. The figure highlights the richness and heterogeneity of the retrieved structures. This allows the data engineer to study the available collections as a whole, progressively refining the targets and the measures. Note that the system allows tailoring the similarity measure on-the-fly; this case uses a domain-specific label-matching function (see Section 4.2.6). In addition to the purely graphical query, it also supports XQuery expressions with the approximate extensions presented in the previous chapter. Figure 6.10 shows a simple query used to group the results by document.

6.3 Concluding remarks

The tools presented in this chapter have been used as a reference implementation to validate the algorithms presented in this thesis, and are also useful as a proof of concept for multi-similarity based XML applications. By themselves, they provide a useful toolkit to the data engineer that needs to develop XML applications using highly tailored and

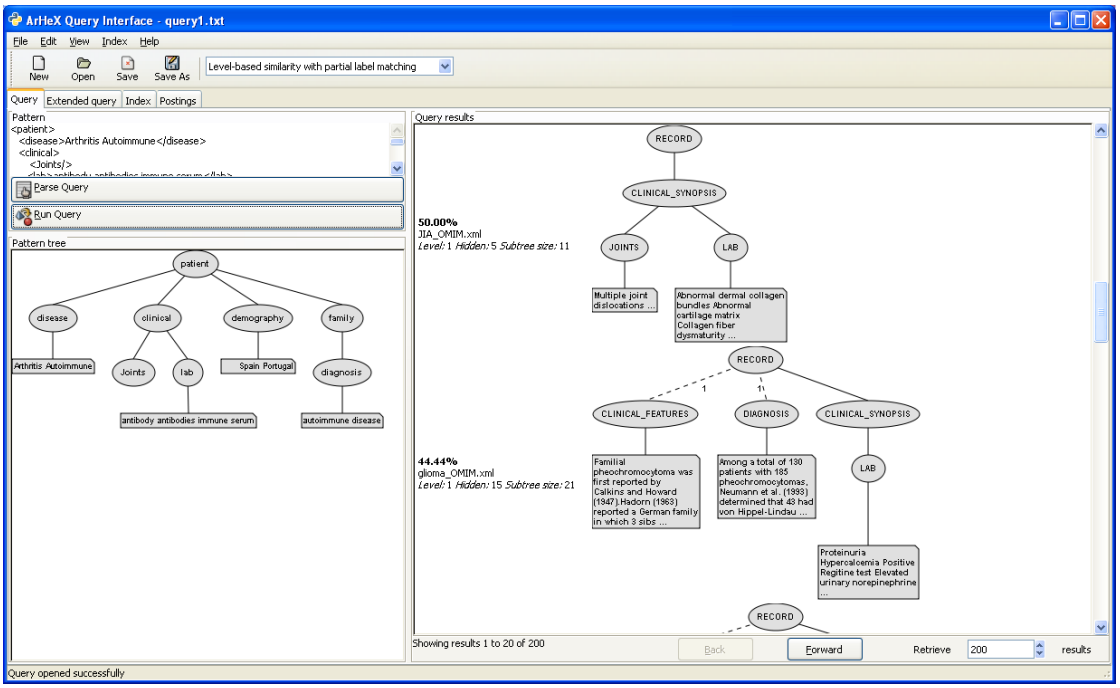


Figure 6.9: Screenshot of the GUI prototype showing some results of a query over a set of heterogeneous biomedical-related collections, using level-based similarity and a flexible label-matching function

flexible similarity measures. In this context, they use in a biomedical research project has been presented as an example.

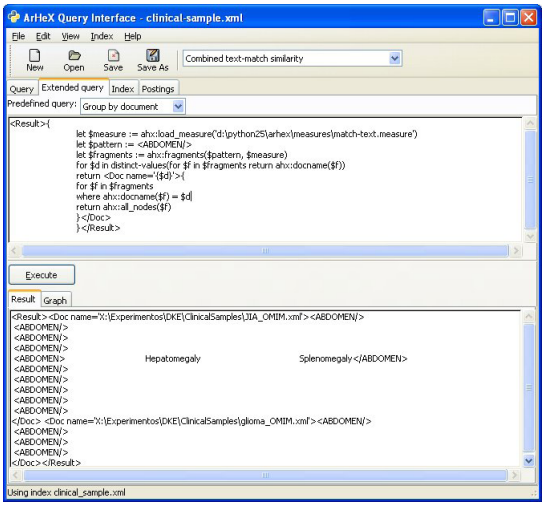


Figure 6.10: XQuery support

Chapter 7

Conclusions

This last chapter presents the main results of the thesis and outlines possible future research lines. It also includes a listing of the publications produced as a result of the work leading to this thesis.

7.1 Summary of results

The core of this thesis is a new model for highly flexible queries over highly heterogeneous XML collections. Such collections have grown in importance as XML has been adopted by new communities that require the treatment of large and complex collections. In many cases, this data does not fit into the traditional categories of "data-centric" and "document-centric" collections; instead, they present significat heterogeneity at many different levels. In particular, their structural heterogeneity makes standard XQuery processing techniques unsuitable.

This motivates a need for highly flexible techniques, that can be applied to collections where not even the stability of parent/child relationship (which is fundamental in XPath and XQuery) can be taken for granted. These techniques are intrinsically approximate, which leads to two additional problems: the definition of similarity measures suitable for XML and the implementation of top-k query processing.

This thesis proposes generic, widely applicable solutions to the issues just outlined, which we review presently. First of all, the thesis proposes formal, quantifiable criteria for defining when a collection is *heterogeneous*. The current literature does not propose such criteria, and describing a collection as "heterogeneous" is generally meant in the limited sense of "collected from several different sources". In this thesis we outline an information-theoretic approach, based on techniques which are well established in other fields but had not been applied to data characterization before.

The thesis also proposes a new, flexible model for XML similarity measures, based on two design goals: the concept of *multi-similarity*, that allows systems that can support many different similarity measures on demand; and the observation that most XML similarity measures are based on a relatively small set of basic measures that are combined and tailored to the problem at hand. As a consequence, our model is based on the *composition* pattern, which allows the creation of new and sophisticated measures by

combining and tailoring smaller building blocks. A method for describing the available measures using a Description Logic-based formalism is also presented.

The core contribution of this thesis is an approach for identifying approximate answers to queries on highly heterogenous XML collections, which allow for processing highly complex and heterogeneous data. The approach is based on the concept of searching for *fragments* in the database based on user-specified *patterns*. Patterns are instances of tree-like structures that represent complex information needs of users. The necessary algorithms are presented, as well as a set of indexing techniques that allow the efficient computation of fragment sets.

The approximate nature of fragment-oriented techniques implies the need for a top-k version of the algorithms. Most existing top-k algorithms for XML are based on approaches that only support *monotonic* similarity measures, which is a strong limitation. In particular, the flexibility of our measure model, in which non-monotonic measures arise very naturally, makes it imperative to address this problem. The thesis presents an approach based on the work of [Xin et al., 2007] in the relational context. The resulting set of techniques represents, to the best of our knowledge, the first general-purpose top-k algorithm for XML which can support non-monotonic measures. We also present an experimental evaluation that shows significant performance gains (between 250% and 350% in our test data set) with respect to the non-top-k version of the algorithm.

The practical applicability of our approach makes it necessary to integrate the algorithms into XQuery-based query processing systems. The thesis presents an external function-based approach for the integration of the algorithms into the language, and outlines the issues that need to be solved in order to integrate the algorithms at the optimizer level. Also for practical reasons, we introduce a representative-based, incremental clustering algorithm for XML that supports flexible similarity measures. This algorithm is particularly well suited for grouping highly complex query results sets in order to facilitate its analysis.

Finally, the proposed techniques have been implemented and tested in ArHeX , which represents the core of a multi-similarity XML application development toolkit. The development of the prototype has allowed us to recognize many practical issues that arise in systems like ours, opening the way for future enhancements.

7.2 Future work

A number of directions for further research have been pointed out throughout the thesis, which we summarize here.

The integration of the techniques presented here in an XQuery system presents several open problems. The three more obvious are *(i)* the integration of approximate algorithms as one more option available to the optimizer; the work of [Michiels et al., 2007] may provide a suitable foundation; *(ii)* integration with proposed extensions of XQuery, in

particular TeXQuery [Amer-Yahia et al., 2004], which provides a natural way to express approximate and top-k queries, although oriented to Information Retrieval; and *(iii)* the possibility of extending the XQuery facilities to define functions, in order to allow the native definition of similarity functions that conform to the model proposed in the thesis. As a first step, the incorporation of the top-k query processing algorithms into an existing, optimizing XQuery implementation such as Galax would provide important insights over the problems involved.

Another contribution that opens new possibilities is the definition of a formal model for the heterogeneity of XML collections. In principle, this makes it possible to define (semi-)automatic techniques to determine automatically what measures can be more appropriate to treat a given collection, in a way analogous to the techniques used by [Bernstein et al., 2005] for combining ontology matching techniques. This approach could simplify the design of multi-similarity applications.

Finally, the work in the development of the prototype has shown that the capability to visualize a heterogeneous XML collection, showing at a glance its regions with distinct characteristics of heterogeneity, would be extremely useful. There is a range of data analysis and visualization techniques that could be applied to the problem.

7.3 Derived publications

This section enumerates the publications that produced this thesis. For each paper we point out the chapters of the thesis that mainly influenced it.

- Ismael Sanz, Marco Mesiti, Giovanna Guerrini and Rafael Berlanga. *Approximate Subtree Identification in Heterogeneous XML Documents Collections Database and XML Technologies*: Third International XML Database Symposium, XSym 2005. Co-located with VLDB 2005, LNCS 3671

- Ismael Sanz, Marco Mesiti, Giovanna Guerrini and Rafael Berlanga. *Highly heterogeneous XML collections: How to retrieve precise results?* 7th Intl. Conf. on Flexible Query answering Systems (FQAS 2006), LNCS 4027

  These two papers provide the foundations of the approximate querying techniques presented in Chapter 4.

- Ismael Sanz, Marco Mesiti, Giovanna Guerrini and Rafael Berlanga: *Fragment-based Approximate Retrieval in Highly Heterogeneous XML Collections.* Accepted for publication in Data & Knowledge Engineering (June 2007)

  This article is a significantly extended version of the two previous papers above.

- Ismael Sanz, Marco Mesiti, Giovanna Guerrini and Rafael Berlanga. *ArHeX: An Approximate Retrieval System for Highly Heterogeneous XML Document Collections.* Demo in EDBT 2006: 10th International Conference on Extending Database Technology, LNCS 3896, 1186–1189

  This paper documents a demonstration of an early version of the prototype presented in chapter 6.

- Ismael Sanz, Juan Manuel Pérez, Rafael Berlanga, and María José Aramburu. *XML Schemata Inference and Evolution.* In 14th International Conference on Database and Expert Systems Applications, DEXA 2003. LNCS 2736.

- Ismael Navas-Delgado, Ismael Sanz, José Francisco Aldana-Montes and Rafael Berlanga: *Automatic Generation of Semantic Fields for Resource Discovery in the Semantic Web.* In 16th International Conference on Database and Expert Systems Applications, DEXA 2005. LNCS 3588.

  These papers present the clustering algorithm from section 5.3, and apply it two different contexts: XML schema inference and ontology matching.

- Ismael Sanz, Rafael Berlanga, Marco Mesiti and Giovanna Guerrini: *Flexible Composition of Indexes and Similarity Measures in XML.* First International Workshop on Ranking in Databases, DBRank 2007. In ICDE 2007 Workshop Proceedings, ISBN 1-4244-0832-6

  This paper presents the foundation of the measures model presented in chapter 3.

- Guerrini, G.; Mesiti, M. & Sanz, I. *An Overview of Similarity Measures for Clustering XML Documents.* In Vakali, A. & Pallis, G. (ed.) Web Data Management Practices: Emerging Techniques and Technologies, Idea Group, 2006, pages 56–78

  This book chapter surveys the state of the art in XML similarity measures. The focus is on clustering, but the framework presented is generic and applicable in different contexts. This publication is the basis for section 2.3.

- Ernesto Jiménez, Rafael Berlanga, Ismael Sanz, Richard McClatchey, Roxana Danger, David Manset, Jordi Paraire and Alfonso Rios: *The Management and Integration of Biomedical Knowledge: Application in the Health-e-Child Project (Position Paper).* In OnToContent'06, 1st International Workshop on Ontology content and evaluation in Enterprise. October 2006. LNCS 4278.

  This paper presents an application of the techniques presented in this thesis in a biomedical research project, which is used as an application scenario in Chapter 6.

Bibliography

Alberto Abelló, Xavier de Palol, and Mohand-Said Hacid. On the midpoint of a set of XML documents. In *16th International Conference on Database and Expert Systems Applications, DEXA 2005*, number 3588 in LNCS, pages 441–450, 2005. doi: 10.1007/11546924_43.

Serge Abiteboul. Querying semi-structured data. In *ICDT '97, 6th International Conference on Database Theory*, volume 1186 of *LNCS*, pages 1–18, 1997.

Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

S. Adalı, P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. A multi-similarity algebra. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 402–413, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-995-5. doi: 10.1145/276304.276340.

Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In James Clifford, Bruce G. Lindsay, and David Maier, editors, *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989*, pages 253–262. ACM Press, 1989.

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 Press. ISBN 1-58113-844-X. doi: 10.1145/988672.988751.

Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava. Tree pattern relaxation. In *Extending Database Technology*, pages 496–513, 2002.

Sihem Amer-Yahia, Nick Koudas, Amélie Marian, Divesh Srivastava, and David Toman. Structure and content scoring for XML. In *VLDB '05: Proceedings of the 31st international conference on Very Large Data Bases*, pages 361–372. VLDB Endowment, 2005.

Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge Unviersity Press, 2003.

Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, 1999.

Denilson Barbosa, Alberto O. Mendelzon, John Keenleyside, and Kelly A. Lyons. ToXgene: An extensible template-based data generator for XML. In *SIGMOD Conference*, 2002.

A. Bernstein, E. Kaufmann, C. Bürki, and M. Klein. How similar is it? Towards personalized similarity measures in ontologies. In *7. Internationale Tagung Wirtschaftsinformatik*, 2005.

Kevin Beyer, Don Chambérlin, Latha S. Colby, Fatma Özcan, Hamid Pirahesh, and Yu Xu. Extending XQuery for analytics. In *SIGMOD'05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 503–514, 2005.

Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language, 2007. URL `http://www.w3.org/TR/xquery/`. W3C Recommendation, 23 January 2007.

Angela Bonifati and Stefano Ceri. Comparative analysis of five XML query languages. *SIGMOD Record*, 29(1):68–79, 2000.

Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0, 1998. URL `http://www.w3.org/TR/1998/REC-xml-19980210`. W3C Recommendation, 10 February 1998.

Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau, and John Cowan. Extensible Markup Language (XML) 1.1 (Second Edition), 2006. URL `http://www.w3.org/TR/2006/REC-xml11-20060816/`. W3C Recommendation, 16 August 2006.

Peter Buneman, Susan B. Davidson, Mary F. Fernandez, and Dan Suciu. Adding structure to unstructured data. In Foto N. Afrati and Phokion Kolaitis, editors, *Database Theory—ICDT'97, 6th International Conference*, volume 1186, pages 336–350, Delphi, Greece, 8–10 1997. Springer.

Barbara Catania and Anna Maddalena. A clustering approach for XML linked documents. In *DEXA '02: Proceedings of the 13th International Workshop on Database and Expert Systems Applications*, pages 121–128, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1668-8.

Don Chamberlin, Jonathan Robie, and Daniela Florescu. Quilt: An XML query language for heterogeneous data sources. In *Proceedings of WebDB 2000 Conference*, 2000.

S. Chawathe, A. Rajaraman, H. García-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493–504, 1996.

Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases*, pages 90–101, Edinburgh, Scotland, U.K., 1999.

Taurai Tapiwa Chinenyanga and Nicholas Kushmerick. An expressive and efficient language for xml information retrieval. *Journal of the American Society for Information Science and Technology*, 53(6):438–453, 2002. ISSN 1532-2882. doi: http://dx.doi.org/10.1002/asi.10057.

V. Christophides, G. Karvounarakis, D. Plexousakis, Michel Scholl, and Sotirios Tourtounis. Optimizing taxonomic semantic web queries using labeling schemes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(2):207–228, February 2004. doi: 10.1016/j.websem.2003.11.001.

Vassilis Christophides, Sophie Cluet, and Jérôme Siméon. On wrapping query languages and efficient XML integration. In *SIGMOD'2000*, 2000.

Chin-Wan Chung, Jun-Ki Min, and Kyuseok Shim. APEX: an adaptive path index for XML data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 121–132. ACM Press, 2002. ISBN 1-58113-497-5. doi: 10.1145/564691.564706.

James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0, 1999. URL `http://www.w3.org/TR/xpath`. W3C Recommendation, 16 November 1999.

William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 201–212, New York, NY, USA, 1998. ACM Press. ISBN 0-89791-995-5. doi: 10.1145/276304.276323.

Gianni Costa, Giuseppe Manco, Riccardo Ortale, and Andrea Tagarelli. A tree-based approach to clustering XML documents by structure. In *PKDD '04: Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 137–148, New York, NY, USA, 2004. Springer-Verlag New York, Inc. ISBN 3-540-23108-0.

Theodore Dalamagas, Tao Cheng, Klaas-Jan Winkel, and Timos Sellis. A methodology for clustering XML documents by structure. *Information Systems*, 31(3):187–228, 2006. doi: 10.1016/j.is.2004.11.009.

Ernesto Damiani and Letizia Tanca. Blind queries to xml data. In *11th International Conference on Database and Expert Systems Applications, DEXA 2000*, 2000.

Giuseppe De Giacomo and Maurizio Lenzerini. Description logics with inverse roles, functional restrictions, and n-ary relations. In *Proceedings of the Fourth European Workshop on Logics in Artificial Intelligence (JELIA'94)*, volume 838 of *LNCS*, pages 332–346. Springer-Verlag, 1994.

A. Deutsch, M. Fernández, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *International World Wide Web Conference*, 1999.

Melvil Dewey. *Dewey decimal classification and relative index*. OCLC Online Computer Library Center, 22nd ed edition, 2003.

P. Dietz and D. Sleator. Two algorithms for maintaining order in a list. In *STOC '87: Proceedings of the nineteenth annual ACM conference on Theory of computing*, pages 365–372, New York, NY, USA, 1987. ACM Press. ISBN 0-89791-221-7. doi: 10.1145/28395.28434.

Hong Hai Do and Erhard Rahm. COMA - a system for flexible combination of schema matching approaches. In *Proc. 28th Intl. Conference on Very Large Databases (VLDB)*, 2002.

Jérôme Euzenat, Philippe Guégan, and Petko Valtchev. OLA in the OAEI 2005 alignment contest. In *Integrating Ontologies*, 2005.

Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Computer and System Sciences*, 66:614–656, 2003.

David C. Fallside and Priscilla Walmsley. Xml schema part 0: Primer second edition, 2004. URL `http://www.w3.org/TR/xmlschema-0/`. W3C Recommendation, 28 October 2004.

Peter Fankhauser. XQuery formal semantics: State and challenges. *SIGMOD Record*, 30 (3):14–19, 2001.

Mary Fernández, Jérôme Siméon, and Philip Wadler. XML query languages: Experiences and exemplars. Communication to the XML Query W3C Working Group, September 1999.

Mary F. Fernández, Jérôme Siméon, Byron Choi, Amélie Marian, and Gargi Sur. Implementing XQuery 1.0: The Galax experience. In *29th Intl. Conference on Very Large Data Bases, VLDB 2003*, pages 1077–1080, 2003.

S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese. Detecting structural similarities between XML documents. In *Proceedings of the Int. Workshop on The Web and Databases (WebDB 2002)*, 2002.

Joerg Freund, Dorin Comaniciu, Yannis Ioannis, Peiya Liu, Richard McClatchey, Edwin Morley-Fletcher, Xavier Pennec, and Giacomo Pongiglione. Health-e-child: An integrated biomedical platform for grid-based paediatric applications. In *4th International HealthGrid conference*, volume 120 of *Studies in Health Technology and Informatics*, 2006.

N. Fuhr and K. Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Proceedings of the 24th Annual International Conference on Research and development in Information Retrieval*, pages 172–180, 2001.

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer D. Widom. *Database System Implementation*. Prentice Hall, 2000.

C. Gokhale, Nitin Gupta, P. Kumar, Laks V. S. Lakshmanan, R. Ng, and B. A. Prakash. Complex group-by queries for XML. In *ICDE 2007*, pages 646–655, 2007.

Torsten Grust. Accelerating XPath location steps. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 109–120, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-497-5. doi: 10.1145/564691.564705.

Giovanna Guerrini, Marco Mesiti, and Ismael Sanz. An overview of similarity measures for clustering XML documents. In Athena Vakali and George Pallis, editors, *Web Data Management Practices: Emerging Techniques and Technologies*, pages 56–78. Idea Group, 2006.

Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD'03*, 2003.

Arnaud Le Hors, Philippe Le Hégaret, Lauren Wood, Gavin Nicol, Jonathan Robie, Mike Champion, and Steve Byrne. Document Object Model (DOM) Level 3 core specification, 2004. W3C Recommendation 07 April 2004.

ISO 8879:1986. Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML), 1986.

A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

Ernesto Jiménez, Rafael Berlanga, Ismael Sanz, Richard McClatchey, Roxana Danger, David Manset, Jordi Paraire, and Alfonso Ríos. The management and integration of biomedical knowledge: Application in the Health-e-Child project (position paper). In

OnToContent'06, 1st International Workshop on Ontology content and evaluation in Enterprise. LNCS 4278., 2006.

Yaron Kanza and Yehoshua Sagiv. Flexible queries over semistructured data. In *PODS '01: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 40–51, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-361-8. doi: 10.1145/375551.375558.

H. Kaplan, T. Milo, and R. Shabo. A comparison of labeling schemes for ancestor queries. In *13th SIAM Symposium on Discrete Algorithms, SODA'02*, 2002.

Raghav Kaushik, Philip Bohannon, Jeffrey F Naughton, and Henry F Korth. Covering indexes for branching path queries. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 133–144, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-497-5. doi: http://doi.acm.org/10.1145/564691.564707.

Gabriella Kazai, Norbert Gövert, Mounia Lalmas, and Norbert Fuhr. The INEX evaluation initiative. In Norbert Fuhr, Hans-Jörg Schek, Gerhard Weikum, Fausto Rabitti, Ralf Schenkel, Norbert Gövert, and Torsten Grabs, editors, *Intelligent Search on XML Data: Applications, Languages, Models, Implementations, and Benchmarks*, volume 2818 of *LNCS*, pages 279–293, 2003.

Pekka Kilpeläinen. *Tree Matching Problems with Applications to Structured Text Databases*. PhD thesis, Dept. of Computer Science, University of Helsinki, 1992.

Pierre P. Lévy. Pixelization paradigm: Outline of a formal approach. In *Pixelization Paradigm*, volume 4370 of *LNCS*, 2007.

Quanzhong Li and Bongki Moon. Indexing and querying xml data for regular path expressions. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 361–370, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-804-4.

Wang Lian, David Wai lok Cheung, 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(1):82–96, 2004. ISSN 1041-4347. doi: 10.1109/TKDE.2004.1264824.

Shaorong Liu, Qinghua Zou, and Wesley W. Chu. Configurable indexing and ranking for XML information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR'04*, pages 88–95, 2004.

W. John MacMullen and Sheila O. Denn. Information problems in molecular biology and bioinformatics. *Journal of the American Society for Information Science and Technology*, 56(5):447–456, 2005. doi: 10.1002/asi.20134.

Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with Cupid. In *The VLDB Journal*, pages 49–58, 2001.

Amélie Marian, Sihem Amer-Yahia, Nick Koudas, and Divesh Srivastava. Adaptive processing of top-k queries in XML. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering (ICDE'05)*, pages 162–173, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2285-8. doi: 10.1109/ICDE.2005.18.

Daniel G. Mcdonald and John Dimmick. The Conceptualization and Measurement of Diversity. *Communication Research*, 30(1):60–79, 2003. doi: 10.1177/0093650202239026.

J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A database management system for semistructured data. *SIGMOD Record*, 26(3):54–66, September 1997.

Wolfgang Meier. eXist: An open source native XML database. In *Web, Web-Services, and Database Systems. NODe, 2002*, volume 2593 of *LNCS*, 2002.

Philippe Michiels, George A. Mihaila, and Jérôme Siméon. Put a tree pattern in your algebra. In *ICDE'07*, 2007.

Tova Milo and Dan Suciu. Index structures for path expressions. In *Proceedings of the 7th International Conference on Database Theory (ICDT)*, pages 277–295, 1999.

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)*, Madison, Wisconsin, USA, June 2002.

Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260, 1995.

František Plašil and Stanislav Visnovsky. Behavior protocols for software components. *IEEE Trans. Softw. Eng.*, 28(11):1056–1076, 2002. ISSN 0098-5589. doi: 10.1109/TSE.2002.1049404.

D. Quass, A. Rajaraman, J.D. Ullman, J. Widom, and Y. Sagiv. Querying semistructured heterogeneous information. *Journal of Systems Integration*, 7(3/4):381–407, 1997.

Praveen Rao and Bongki Moon. PRIX: Indexing and querying XML using Prüfer sequences. In *ICDE'04: Proceedings of the 20th International Conference on Data*

Engineering, page 288, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2065-0.

J. Robie, J. Lapp, and D. Schach. XML query language (XQL), 1988. URL `http://www.w3.org/TandS/QL/QL98/pp/xql.html`. September 1998.

P.J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational Applications in Math*, 20:53–65, 1987.

M. Salicrú, S. Vives, and J. Oca na. Testing the homogeneity of diversity measures: a general framework. *Journal of Statistical Planning and Inference*, 132:117–129, 2005.

Gerard Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

Ismael Sanz, Juan Manuel Pérez, Rafael Berlanga, and María José Aramburu. XML schemata inference and evolution. In *Database and Expert Systems Applications, 14th International Conference, DEXA 2003*, pages 109–118, 2003.

Torsten Schlieder. Similarity search in XML data using cost-based query transformations. In *Proceedings of SIGMOD WebDB Workshop*, 2001.

Torsten Schlieder. Schema-driven evaluation of approximate tree-pattern queries. In *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*, pages 514–532, London, UK, 2002. Springer-Verlag. ISBN 3-540-43324-4.

Torsten Schlieder and Felix Naumann. Approximate tree embedding for querying XML data. In *ACM SIGIR Workshop On XML and Information Retrieval*, 2000.

S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6:184–186, December 1977.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

Dennis Shasha, Jason T. L. Wang, Huiyuan Shan, and Kaizhong Zhang. ATreeGrep: Approximate searching in unordered trees. In *14th International Conference on Scientific and Statistical Database Management (SSDBM'02)*, pages 89–98, 2002.

David Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 2003.

Edward H. Simpson. Measurement of diversity. *Nature*, 163:688, 1949.

Jay D. Teachman. Analysis of Population Diversity: Measures of Qualitative Variation. *Sociological Methods Research*, 8(3):341–362, 1980. doi: 10.1177/004912418000800305.

Anja Theobald and Gerhard Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*, pages 477–495, London, UK, 2002. Springer-Verlag. ISBN 3-540-43324-4.

Martin Theobald. *TopX - Efficient & Versatile Top-k Query Processing for Text, Structured, and Semistructured Data.* PhD thesis, Universität des Saarlandes, 2006.

W3C HTML Working Group. XHTML™ 1.0, The Extensible HyperText Markup Language (Second Edition), 2002. URL `http://www.w3.org/TR/xhtml1`. W3C Recommendation, 26 January 2000, revised 1 August 2002.

R.A. Wagner and M.J.Fisher. The string to string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. ViST: A dynamic index method for querying XML data by tree structures. In *SIGMOD*, 2003.

Jennifer Widom. Data management for XML: Research directions. *IEEE Data Engineering Bulletin*, 22(3):44–52, 1999.

N. Wirth. Type extensions. *ACM Trans. Program. Lang. Syst.*, 10(2):204–214, 1988. ISSN 0164-0925. doi: 10.1145/42190.46167.

Kam-Fai Wong, Jeffrey Xu Yu, and Nan Tang. Answering XML queries using path-based indexes: A survey. *World Wide Web*, 9(3):277–299, October 2006. doi: 10.1007/s11280-006-8557-z.

Dong Xin, Jiawei Han, and Kevin Chen-Chuan Chang. Progressive and selective merge: Computing top-k with ad-hoc ranking functions. In *SIGMOD'07*, 2007.

Rui Xu and II Wunsch, D. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

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 Press. ISBN 1-59593-060-4. doi: 10.1145/1066157.1066243.

Jong P. Yoon, Vijay Raghavan, and Venu Chakilam. Bitmap indexing-based clustering and retrieval of XML documents. In *ACM SIGIR'01 Workshop on Mathematical/Formal Methods in IR*, 2001.

K. Zhang, R. Stgatman, and D. Shasha. Simple fast algorithm for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):245–1262, 1989.

Colophon

This thesis was edited using LyX 1.5 and TeXnicCenter, and typeset using the MiKTeX 2.5 distribution of LaTeX 2_ε. The document class is `scrrprt`, the KOMA-Script version of the traditional `report` class. The additional packages are pretty standard, except maybe for `microtype`, which subtly enhances the overall look of the document using "micro-typographic" techniques which I certainly cannot claim to understand. The BibTeX bibliography was maintained using JabRef 2.1, and was typeset in the `natbib` author-year style.

Raw experimental results where turned into graphs using the `R` 2.4 statistical software. Additional illustrations where created using the GraphViz suite, and either used directly as EPS files or transformed into LaTeX-readable PGF/Ti*k*Z pictures using `dot2tex`. Figures 3.1, 3.2 and 6.1 were produced using Microsoft® Visio® 2003. All screenshots were captured and converted using GIMP and ImageMagick®.