Chapter 1

Introduction

With the advent of the Internet — and particularly of the World Wide Web — the traditional distinction between regularly structured data (as stored in databases) and unstructured data (as stored in text files) has become blurred. Data on the Internet often has an irregular, partial, and implicit structure. The eXtensible Markup Language (XML) has been designed to provide a common basis not only for the representation of such *semistructured data*, but also of regularly structured and even unstructured data. Three key concepts enable this language to represent such different types of data: First, XML is a *generic* markup language — sometimes called a meta markup language — with a concise syntax. It is easy to define a concrete language based on the generic one to represent and exchange data of a certain kind. Second, XML allows an *integrated* representation of data and its schema. This property makes it possible to model the regular parts of the logical structure of the data, but also allows for the representation of variations and irregularities. Third, XML documents have a *self-describing* structure. This is a consequence of the requirements of the XML specification, which states that the structure should be "human-legible and reasonably clear" [BPSM00].

1.1 Searching in XML Data: Why Traditional Approaches Fail

How can the self-describing structure contribute to improving the information search in XML data? We believe that the structure can help to yield more *precise* queries — but the structure should only serve as a *guide* to locate the desired information. Neither retrieval models developed for text data, nor the currently established XML query languages meet these requirements, as we will now discuss.

```
<catalog>
  ...
  <cd>
    <category>Classics</category>
    <performer>Rachmaninov</performer>
    <title>A window in time</title>
    <tracks>
      <track>
        <title>Piano sonata</title>
      </track>
      ...
    </tracks>
  </cd>
  ...
</catalog>
```

```
<catalog>
  ...
  <mc>
    <performer>Ashkenazy</performer>
    <composer>Rachmaninov</composer>
    <title>Piano concerto no. 1</title>
    <tracks>
      <track>
        <title>Vivace</title>
      </track>
      ...
    </tracks>
  </mc>
  ...
</catalog>
```

Figure 1.1: Two XML documents containing information about media products.

Traditional keyword-based retrieval models are not designed to incorporate the structure of XML documents, and are thus forced to ignore it. However, ignoring the structure means ignoring important information describing the content to be searched, which leads to less precise queries than possible. A further limitation of text retrieval models is the use of whole documents as results. This is not appropriate for XML, because users are typically not interested in a whole document, but rather in the elements contained within it. Moreover, the requested elements may appear at any level in the hierarchical structure of the document.

Example: Consider a media store selling books, videos, and sound storage media. Figure 1.1 shows two XML documents with product information. A user may be interested in a CD with piano concertos by Rachmaninov, and phrases the keyword query "piano concerto rachmaninov". If no result elements have been specified, then the query retrieves two catalogs with possibly thousands of objects. Otherwise, if the administrator has defined CDs and MCs as possible results, then it retrieves all `cd` and `mc` elements that contain at least one of the terms "piano", "concerto", and "Rachmaninov". However, the users cannot specify that they prefer CDs over MCs, or that they prefer media with the title "piano concerto" over media containing tracks with the title "piano concerto". Similarly, the users cannot specify their preferences for the composer Rachmaninov over the performer Rachmaninov, despite the presence of all of the necessary information in the catalog.

XML query languages developed by the database community incorporate the structure of the documents (see [FSW99, BC00] for surveys). These languages are well suited for applications that query and transform XML documents — but they are not appropriate for non-expert users. It is hard for the users to formulate queries, not only because of the complexity of the query syntax, but also because the users are required to have thorough knowledge of the structure of the documents being queried. This knowledge is particularly hard to acquire if the documents have a large, heterogeneous structure. The interpretation of a query is rigid: A substructure of a document either matches the query or it does not. This semantics is often not satisfactory: In many cases, the users only partially know the structure of the documents, and therefore formulate queries with an incomplete data model in mind. However, even if they know the complete structure, they can never have complete knowledge of the content. The deviations between the users' knowledge and the actual structure and content of the documents may lead to several cases where the query fails, even though answers that may interest the users exist. Typical cases are:

- The query only partially matches a relevant document.
- The structure of the query differs from the structure of a relevant document.
- The requested content exists, but not in the structural context specified by the query.
- The requested content does not exist, but there is semantically related content.

Note that language constructs like regular path expressions do not solve these problems, because the users must know that structural heterogeneity exists, and what the regular expression must look like in order to match the requested results.

Example: Consider the two documents in the media store shown in Figure 1.1, and assume that the store is able to answer queries formulated in the language XQL [RLS98]. The query

```
/catalog/cd[title="Piano concerto" and composer="Rachmaninov"]
```

will neither retrieve CDs with a track title "Piano concerto", nor CDs of the category "Piano concerto", nor concertos performed by "Rachmaninov", nor sound storage media other than CDs with the appropriate information. The query will also not retrieve CDs where only one of the keywords "Piano" or "concerto" appears in the title. It will fail if the hierarchical relationships between `cd` elements and `composer` elements are not consistent within the documents in the media store. Of course, the users can phrase a query that exactly matches the cases mentioned — but they must know beforehand that such similar results exist, and how they are represented. Moreover, because all of the results of the redefined query are treated equally, the users cannot express their preferences, and receive the results in an arbitrary order.

The examples demonstrate that there is a strong need for a search method that incorporates the structure of the documents, supports the retrieval not only of exact matches, but also of results similar to the query, valuates the similarity between the query and the results, and ranks the results by decreasing similarity.

1.2 Thesis Objective

The objective of this thesis is to develop and describe a novel approach to search in XML data with complex, heterogeneous structure. This includes the design of a simple yet expressive query language and the development of efficient algorithms for the evaluation of queries. The language and algorithms must meet the following requirements:

- Query formulation must be simple even for occasional users, and must require only partial knowledge about the collection to be queried.
- The language must support expressive queries that specify selection conditions on both the content and the hierarchical structure of XML documents.
- The interpretation of a query must be vague to find not only exact matches, but also results with a structure and content similar to the selection conditions of the query.
- The results retrieved by a query must be ranked by decreasing similarity to the query.
- The basic parameters of the similarity measure must be variable, so that they can be adapted to the characteristics of various document collections.
- A query must be evaluated in polynomial — typically sublinear — time with respect to the size of the collection. In particular, the best n results must be retrieved efficiently.

1.3 Solutions Presented in the Thesis: An Overview

In this thesis, we propose an innovative method for fast similarity search in XML data, which uses the self-describing structure of XML documents as a guide to locate the information a user is interested in. We introduce a new query language, approXQL, with a user-friendly syntax and a similarity-based semantics, develop a complete algorithmic framework to evaluate approXQL queries, describe our prototypical implementation of a query processor, and discuss the results of a thorough efficiency analysis.

The syntax of approXQL is simple but expressive. It allows to formulate queries that make use of the hierarchical structure of XML documents. The design of the syntax is based on the assumption that typical users have only partial knowledge of the structure of the documents, and are not able or willing to cope with a complex syntax. To formulate a query, the users only need to know the names of the XML elements and attributes they are interested in.

The semantics of approXQL is designed for the retrieval of exact matches, but also of results similar to the query. We view the similarity between an approXQL query and an XML document as the degree of deviation between the content and structure requested by the query and the content and structure actually existing in the document. The fewer deviations exist, the more similar the query and the document are considered to be, and the higher the position of the document within the list of results.

In order to formally define our notion of similarity, we use trees to model both queries and documents in a uniform way. We show how a collection of XML documents can be interpreted as a single data tree. Based on this interpretation, we are able to shift from the static notion of XML documents to the flexible concept of logical documents as subtrees of the data tree. We also demonstrate how approXQL queries can be decomposed into a set of conjunctive queries, and how each conjunctive query can be mapped to a query tree.

For each logical document in the data tree, we try to find exact embeddings of the query trees. An exact embedding is a mapping of the nodes in a query tree to the nodes in the document such that the mapping preserves the types and values of the nodes, as well as the parent-child relationships between the nodes. Not for all logical documents semantically close to the query exact embeddings are possible. Therefore, we propose the theoretical concept of a query closure. The closure consists of all query trees that can be derived from the original query trees via transformation sequences. We allow the deletion, permutation, and insertion of query nodes, as well as the change of the values assigned to the nodes. Each basic transformation has a cost, which is defined by a domain expert. The total cost of a transformation sequence determines the embedding cost of the resulting query tree. For each logical document, we select all transformed query trees that can be exactly embedded, and choose the one with the lowest embedding cost. The cost of this tree defines the distance between the original query and the document. A pair consisting of a logical document and its cost forms a result. The results are sorted by increasing cost and displayed to the users.

The theoretical model of approximative embeddings implies a naive query-evaluation method. However, this method is very inefficient because it assumes that the query trees are transformed and embedded independently of each other. Therefore, we present an integrated, modular, and performant algorithmic framework for evaluating queries. To establish the connection between the theoretical model and its implementation, we introduce the expanded representation of a query, which is a compact data structure that encodes all query trees that can be derived from the query via transformation sequences. The expanded representation is the starting point for the construction of a query-execution plan, which consists of operators that work on sets of data-tree nodes. With the help of a query-execution plan, we can view the evaluation of a query as an optimization problem, where for each logical document the optimal solution (the lowest-cost embedding) must be found. A query-execution plan is evaluated bottom-up. For all subtrees of the query trees and for all subtrees of the logical documents, the optimal solutions are computed. Subsequent operators combine these solutions to optimal solutions for the next level, until the top-level operator is reached.

Usually, users are interested only in the best n results. To find those results without computing similarity scores for all logical documents in the collection, we analyze the relationships between a data tree and its path tree (schema). We show that the schema can be used to estimate the best k transformed query trees. To select these trees, we make use of our query-evaluation framework. We modify the plan operators so that they can select the best k embeddings per logical document, instead of selecting only the best one. The selected query trees are then used as "second-level" queries. They are sorted by embedding cost, and are then successively executed against the data tree. Because there is no fixed relationship between k and n, we propose an incremental algorithm that guesses an initial value for k. If the first pass retrieves less than n results, the algorithm increases k and starts a new pass. The evaluation stops if at least n results are found.

1.4 Thesis Outline

We outline the thesis by summarizing each chapter.

Chapter 2: State of the Art. We provide a thorough analysis of related work. In particular, we review XML query languages, flexible query mappings, information retrieval models for XML, and distance measures for labeled trees. At the end of the chapter, we give a taxonomy of the reviewed approaches, and discuss why none of them fulfills our requirements.

Chapter 3: The approXQL Query Language. This chapter provides a tutorial-style introduction to the approXQL query language. We present a simple yet expressive core syntax, which allows even unexperienced users to formulate queries that include selection conditions on the structure of XML documents. For seasoned users, we define an extended syntax that allows the modification of the default semantics of approXQL, and the specification of typed selection conditions.

Chapter 4: Modeling Documents and Queries. We propose type-value trees as a means to model XML documents and approXQL queries in a uniform way. A collection of XML documents is modeled as a single type-value tree, where each subtree is considered to be a logical document. An approXQL query is modeled as a set of type-value trees, where the nodes in the trees are enriched by selection predicates.

Chapter 5: Querying by Approximate Tree Embedding. This chapter formally defines the semantics of approXQL. We map the problem of finding exact results for a query to the problem of finding embeddings of query trees within the data tree. To find results similar to the query, we propose a novel similarity measure that is based on query transformations. The total cost of a sequence of transformations measures the similarity between the original query and each logical document in the data tree.

Chapter 6: Direct Query Evaluation. We propose query-execution plans for the efficient evaluation of approXQL queries. A plan consists of operators that perform operations on sets of data-tree nodes, and in parallel calculate the transformation costs. We prove that during the evaluation of a plan all query results (and only those) are found, and that the calculated costs are correct. We also propose techniques to optimize the evaluation of a plan and analyze the upper bound for the number of operators in a plan.

Chapter 7: Schema-Driven Query Evaluation. This chapter addresses the efficient retrieval of the best n query results. We show that the path tree (schema) of a data tree can be used to estimate the best k transformed query trees, which in turn are executed against the data tree to find the best n results. To select the transformed trees, we use query-execution plans with slightly modified operators. We propose an optimization technique to reduce the number of transformed trees necessary to find the best n results.

Chapter 8: Efficient Algorithms for Plan Operators. In this chapter, we establish the connection between the formal definitions of the operators used in query-execution plans and their actual implementations. We extend a well-known tree-numbering scheme so that it allows to determine the sum of insertion costs of nodes on a path, without actually traversing the path. Using this numbering scheme, we present algorithms for the operators that have time and space complexities linear in the size of the inputs.

Chapter 9: The approXQL Query Engine. This chapter introduces the prototypical implementation of an approXQL query engine. The engine implements both the direct and the schema-based query-evaluation methods. Furthermore, it provides a graphical query editor and consists of all modules necessary to load, index, and display XML documents.

Chapter 10: Experimental Efficiency Analysis. We present the results of extensive tests to evaluate the efficiency of the query engine using collections of real and synthetic XML documents. In particular, we show the dependencies of our algorithms on several query and data parameters.

Chapter 11: Conclusion. In this chapter, we summarize the main contributions of the thesis. We also point out open problems and interesting directions for future work.

.