Chapter 11

Conclusion

In this thesis, we proposed an innovative approach for fast similarity search in XML data — with a particular focus on data with a complex, heterogeneous structure. We introduced approXQL, which is the first XML query language whose syntax makes use of the hierarchical structure of XML data, and whose semantics allows a vague interpretation not only of the content-selecting query parts, but also of the structure-selecting parts. The following sections summarize the main contributions of the thesis, and point out open problems and interesting directions for future work.

11.1 Summary of Contributions

We summarize the contributions of this thesis with respect to the syntax and semantics of approXQL, to the query-evaluation algorithms, and to the prototypical implementation.

Syntax

We proposed a user-friendly syntax for approXQL. It consists of two parts: a core syntax for unexperienced users, and an extended syntax for users who are more familiar with the semantics of the language and with the structure and content of XML data.

The core syntax is simple but expressive. It allows the formulation of Boolean queries that specify conditions on both the content and hierarchical structure of XML documents. Its design was motivated by the observation that typical users are not willing or able to learn all of the details of the sometimes complex and heterogeneous structure of XML documents.

To phrase a query using the core syntax, users only need to know the names of elements or attributes they are interested in.

The extended syntax provides language primitives that allow the users to modify the default semantics of the language by relaxing or restricting query transformations. It also supports a type system, which takes data properties ignored by the core syntax into account.

Semantics

We presented an innovative semantics for approXQL, which aims at bridging the gap between the expressiveness of structured queries and the vagueness of information retrieval. With this semantics, approXQL is well suited for similarity search in XML data with a complex, heterogeneous structure.

For the formal definition of the semantics, we proposed type-value trees as a means to model XML data and approXQL queries in a uniform way. By interpreting a collection of XML documents as single data tree, we were able to shift from the notion of physical XML documents to a more flexible notion that considers each subtree of the data tree as logical document.

Based on the tree interpretation of queries and document collections, we proposed transformations of query trees as a powerful means for finding results similar to the original query. The semantics supports four types of basic transformations: the deletion of query nodes, the insertion of nodes into query trees, the changing of values assigned to the leaves and inner nodes in query trees, and the permutation of hierarchical query relationships.

To determine the similarity between a query and the logical documents selected as results, we proposed a flexible cost model: Each single transformation of a query tree is annotated with a cost. The total cost of a sequence of transformations applied to a query tree measures the similarity between the original query and a particular logical document. The logical documents selected by the query are ranked by decreasing similarity. By adjusting the costs of the transformations, the semantics of approXQL can be tailored to the needs of different users and to the varied characteristics of XML documents.

Algorithms

The algorithmic framework establishes the connection between the theoretical model of the semantics and its implementation as part of a query processor. We succeeded in developing a suite of algorithms that overcome the inherent exponential time behavior of the theoretical model caused by the infinite number of possible transformation sequences for query trees.

The algorithms evaluate a query in polynomial time with respect to the size of the query and the data tree. In practical cases, the required time is even sublinear, because the complexity of the algorithms essentially depends on the number of times the elements, attributes, and words requested by the query occur in the data tree.

To avoid the explicit creation of the closure of transformed query trees, we introduced the expanded representation of a query. The expanded representation is a compact tree that implicitly includes all query trees that can be derived from the original query via transformation sequences.

An expanded query representation does not only establish the connection to the theoretical model of query evaluation, but is also the starting point for the construction of a query-execution plan. As a basis for query-execution plans, we proposed a novel algebra. The operators of the algebra perform operations on sets of data-tree nodes, and in parallel calculate the embedding costs of the query trees. Query-execution plans are modular (because they consist of independent operators), extensible (because new language constructs can be realized using the operators), and implementation independent (because the operators are defined declaratively).

We presented two techniques to optimize the evaluation of query-execution plans: First, we investigated equivalences between algebra operators, and showed how query-execution plans can be compacted using these rules. Second, we adopted the dynamic programming principle to avoid the repeated evaluation of shared subplans.

To find the best n results without computing similarity scores for all documents in the collection, we analyzed the relationships between a data tree and its path tree (schema). We showed that the schema can be used to estimate the best k transformed query trees. To select these trees, we made use of the query-evaluation framework developed so far. We only modified the operators in a plan so that they select the best k embeddings per logical document, instead of only the best one. The constructed query trees are sorted by embedding cost, and successively executed against the data tree until the best n results are found.

We presented the hybrid query-evaluation method as a variant of the schema-driven query-evaluation method. It uses an optimization technique that effectively reduces the number of transformed trees necessary for finding the best n results. The reduction is achieved by detecting common parts in query trees, and by merging trees with common parts.

The operators of the algebra are defined declaratively. To use them in a query processor, we presented efficient algorithms that make use of the topological relationships between data-tree nodes. We extended a well-known tree-numbering scheme so that is allows to determine

how many nodes must be inserted into a query tree to enable an exact matching. With this numbering scheme, we can calculate the total insert cost of all nodes to be inserted into the query tree in constant time — without actually inserting them. We showed that, based on this numbering scheme, all operators can be implemented by algorithms with time and space complexities linear in the size of the inputs.

We proposed a general method for the use of index structures within our algorithmic framework. We concentrated on the adaptation of inverted indexes, but pointed out that all indexes for one-dimensional access paths like B-trees and suffix trees can be used.

Prototype

We introduced our prototypical implementation of an approXQL query engine. The engine supports the direct query-evaluation method and both variants of the schema-driven query-evaluation method. Furthermore, the prototype consists of all modules necessary to load, index, and display XML documents. It also provides a graphical query editor.

To evaluate the efficiency of the query engine and to compare the response times achieved by the three evaluation methods, we carried out extensive tests using collections of real and synthetic XML documents. We found out that the two variants of schema-driven query evaluation outperform the direct method in most cases. Furthermore, we saw that the hybrid query-evaluation method is usually superior to the non-optimized variant of schema-driven query evaluation.

The prototype demonstrates that the implementation of our model for similarity search in XML data is feasible, and that our algorithmic framework is efficient enough to be used in a real-life query engine.

11.2 Future Work

In this section, we point out open problems and suggest interesting directions for future work.

Estimation of Basic Costs

The costs for basic transformations are the free parameters of the approXQL semantics. They allow the adaptation of the model to a wide range of distinct types of documents. Presently, a domain expert must estimate the basic costs for a collection by trial and error. Future

work should therefore include the development of a methodology for estimating the costs. This may include the support of an expert by a set of rules, the use of relevance feedback to adapt the basic costs to the relevance valuations carried out by the users, and techniques to detect semantic relationships between the contents of elements in order to find appropriate value-change and insertion costs. The recently developed methods for estimating the similarity between XML documents reported in [Lee02, NJ02] may form the basis for future investigations.

Negation

Negation is not part of the current approXQL specification because its non-standard semantics would require a non-standard definition of a negated expression. In a standard semantics, a negated subquery would exclude all logical documents from the result set that have no occurrences of that subquery in the context defined by the enclosing query. Unfortunately, we cannot apply this semantics to queries for which transformations are permitted: A transformation within the context of a subquery enlarges the closure of that subquery, and also the closure of the whole query. Consequently, the number of possible query embeddings increases, and therefore the number of results increases as well. If the subquery is negated, then that effect is reversed: A transformation would enlarge the closure of that subquery — but it would *reduce* the size of the closure of the whole query, because more query subtrees where excluded than before. Future work may include the development of a semantics that allows query transformation for non-negated subqueries only.

Graph-structured Data

The current specification of the approXQL semantics builds on tree-structured data. This is a simplified view, because XML documents may contain element references (the ID/IDREF mechanism), or links to elements either in the same or in another document (the W3C recommendation XLink [DMO01]). The usual representation form for this type of document collection is a directed labeled graph with a unique root [ABS99]. The extension of our approach to graph-structured data would be useful, but also very challenging. The most problematic part is the implementation of an ancestor-descendant join, because we need a method that allows us to test (i) whether two nodes in the graph are connected by a path, and (ii) which one of the detected paths is the shortest. A promising direction for future work is to investigate the usability of *hub indexes* [GSVGM98] for that task.

Term Weights

The interpretation of approXQL queries relies on the approximate query-matching distance, which penalizes for the differences between the structure of the query and the structure of the results. As a consequence, equally structured results get the same score. For text-rich XML documents, it might be useful to incorporate the distribution of terms in the calculation of the scores: Results for which query terms are well descriptors should get a higher score than results that are less good described. The descriptiveness of a term could be estimated using a variant of the classical tf*idf formula [SM83]. However, the definition of the term weights is not obvious. In text documents, term frequencies and inverse document frequencies (the main constituents of term weights) are calculated with respect to static document boundaries. In XML documents, however, terms appear within elements. A term describes not only the content of its enclosing element, but also the content of elements at higher hierarchy levels. This makes it necessary to adapt a classical weighting scheme to the requirements of XML, or to develop a completely new weighting scheme.

Ordered Embeddings

The embedding function, which maps nodes in a query tree to nodes in a data tree, does not require that sibling nodes in the query tree have the same left-to-right order as their matches in the data tree. Sometimes however, (partially) ordered embeddings would be useful: The users could specify that two tracks of a CD must be in a certain order, or that some words must form a sequence (with interspersed other words). The partial horizontal order of embeddings cannot be implemented when using the current definition of query-execution plans, because the ordering constraint does not allow an independent evaluation of the operators. To overcome this problem, query-execution plans must be extended by a flexible mechanism to define and check inter-operator constraints.

Horizontal Distances

The current specification of the approXQL semantics defines insertions of nodes into a query tree as a means of bridging vertical distances in the data tree. However, the horizontal distance between two subtrees of the data tree may also be interesting: The closer two sections in a text, the more related their contents probably are; the closer two words, the higher the probability that they form a semantic unit. The integration of horizontal distances into the approXQL framework raises two yet unsolved problems: the definition of a measure that incorporates horizontal distances, and the development of efficient algorithms for the evaluation of a query based on that measure.