**Dieses Dokument ist eine Zweitveröffentlichung (Postprint) /**

**This is a self-archiving document (accepted version):**

Katja Seidler, Eric Peukert, Gregor Hackenbroich, Wolfgang Lehner

# Approximate Query Answering and Result Refinement on XML Data

**Erstveröffentlichung in / First published in:**

*Scientific and Statistical Database Management: 22nd International Conference.* Heidelberg, 30.06. - 02.07.2010. Springer, S. 78–86. ISBN 978-3-642-13818-8.

DOI: http://dx.doi.org/10.1007/978-3-642-13818-8_7

Diese Version ist verfügbar / This version is available on:

https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-829944

SLUB
Wir führen Wissen.

TECHNISCHE UNIVERSITÄT DRESDEN

QUCOSA Quality Content of Saxony

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

# Approximate Query Answering and Result Refinement on XML Data

Katja Seidler[1], Eric Peukert[1], Gregor Hackenbroich[1], and Wolfgang Lehner[2]

[1] SAP Research CEC Dresden, Chemnitzer Str. 48, 01187 Dresden, Germany
{firstname.lastname}@sap.com
[2] Technische Universität Dresden, 01062 Dresden, Germany
wolfgang.lehner@tu-dresden.de

**Abstract.** Today, many economic decisions are based on the fast analysis of XML data. Yet, the time to process analytical XML queries is typically high. Although current XML techniques focus on the optimization of query processing, none of these support early approximate feedback as possible in relational Online Aggregation systems.

In this paper, we introduce a system that provides fast estimates to XML aggregation queries. While processing, these estimates and the assigned confidence bounds are constantly improving. In our evaluation, we show that without significantly increasing the overall execution time our system returns accurate guesses of the final answer long before traditional systems are able to produce output.

## 1 Introduction

The data volume and growing rates of today's business systems make approximate query processing an inevitable technique for fast analyses. Online Aggregation (OLA) has been proposed as an approach for analytical processing of relational data. In OLA, the database system quickly returns approximate answers to aggregation queries together with statistical guarantees on the error bounds. This computing paradigm allows users to more flexibly explore data: Query processing may be terminated at any time once sufficient accuracy has been reached; if exact answers are needed, they can be computed with little overhead as compared to traditional systems.

In this paper, we show how OLA can be performed on XML data. We present a novel query processing system—referred to as XML Database Online (XDBO) System—which performs OLA on XML data in a scalable way. We have been faced with the following main challenges: Query processing on XML data is heavily dominated by the evaluation of so called twig patterns. This pattern matching process involves a multitude of structural joins and is a major bottleneck within XML query processing. Compared to relational databases the queries differ in both the number and the kind of joins. Additionally, in current XML processing systems an aggregate cannot be returned until the whole structural join operation is completed. This is a time-consuming task, especially for complex queries and/or queries over large data sets. We address these challenges and make the following contributions:



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

- We propose novel operators for the random and non-blocking selection and join of query path patterns.
- We show how sideways information passing can be used for fast approximate query answering and introduce the architecture of the XDBO System.
- With an extensive evaluation of a prototypical implementation we demonstrate the feasibility and the efficiency of our approach.

Furthermore, we point out optimization possibilities and present our prototype in the long version of this paper [10].

## 2 Indexing and Pattern Matching

In this section, we introduce the foundations of our proposed XDBO System that are based on the following requirements of OLA: First, for a scalable query processing and for fast first answers the pattern matching must be performed in a *non-blocking fashion*, and second, to guarantee statistical valid estimates and error bounds XML elements have to be processed in *random order*. We now describe two novel pattern matching operators and a special index structure that are designed to meet the given requirements. The operators reflect the general procedure of the approximate query processing in the XDBO System: Instead of processing a query pattern as a whole, we decompose it into a set of query path patterns. We identify all path patterns of the query and remember the positions of branching nodes that connect individual paths. With a novel selection operator (Section 2.2) we search for solutions to query path patterns; a special index structure (Section 2.1) facilitates its efficiency. Finally, a join operator (Section 2.3) connects the solutions of individual path patterns.

Figure 1 illustrates the main steps, starting from the query pattern, all identified query path patterns and the constructed execution plan for a count query over the XPath expression `//a[.//b[c]/d]//e[f]/g`.

### 2.1 Element Path Index

Traditional XML database systems utilize special numbering schemes to speed up query processing: They label all elements of an XML document, often encode the structural relationship into the labels and store them for each element into an index structure. The index is then used to accelerate a pattern matching operation. To support pattern matching in a system that meets the OLA requirements a special index is needed. Based on a thorough state-of-the-art analysis, we decided to utilize the extended Dewey numbering (EDN) scheme [7]. In addition to the encoding of the structural relationship, the EDN scheme encodes—with the help of a Finite State Transducer (FST)—the whole root-to-node path of each element into the label. Therefore, only the labels of the leaf node element of a query pattern need to be accessed during pattern matching. The solutions of the query pattern $//a//b/c$ can be found by examining all labels of $c$ elements and check if they match the query pattern. Hence, only the decoding of EDN labels and the comparison of the retrieved path patterns with the query path pattern have to be done. This enormously speeds up and simplifies pattern matching.



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

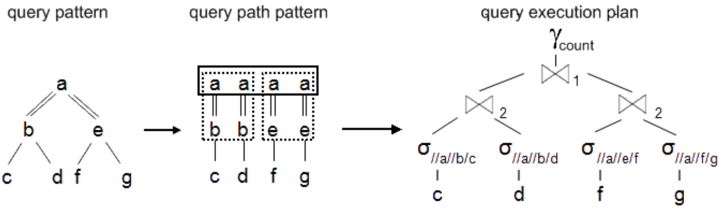


**Fig. 1.** Generation of the execution plan

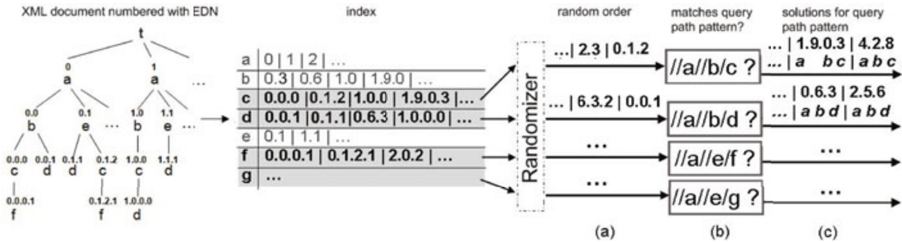


**Fig. 2.** Selection process

### 2.2 Path Pattern Selection Operator

Based on an index that stores EDN labels for each element type, we propose a novel selection operator $\sigma_{XPath}$ (see Figure 1). The characteristic of this operator is the processing of input elements—the EDN labels from the index—in random order. On the right part of Figure 2, we show how this is achieved (the XML document and the corresponding index are given in the left part): The element labels of the leaf nodes of the query are extracted from the index in random order (a). If the index structure does not guarantee randomness a randomizer has to be used. During the selection process (b) the operator translates each label of the random input stream with the FST into a root-to-node path and compares it with the query path pattern. If there is a match a solution is found. For further processing, not only the selected labels but also the positions of the query path pattern elements are memorized (c).

### 2.3 Path Pattern Join Operator

To find solutions for the whole query pattern the results of the individual query path patterns need to be joined at branching nodes. Branching nodes (specifically their position *pos* in the path pattern) are defined by the join operator $\bowtie_{pos}$ (see Figure 1). The join process is much more complex than the relational join operation. Rather than comparing columns of different tuples positions of labels are compared to join the records. Two labels can only be joined if they coincide from root position up to the position of the associated branching node.



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

# 3 Query Processing

This section describes how the XDBO System effectively uses the presented index and the two pattern matching operations to provide early estimates of the final aggregate of a query. To achieve high scalability we employ the concept of sideways information passing. According to this concept, operations at a single level of a query plan are performed concurrently while allowing to share some of its intermediate results with other operations at the same level. Thereby, preliminary result tuples can be generated on the fly on each layer of the query plan and are used to provide an estimate for the final query answer. Sideways information passing was introduced as a design paradigm for OLA of relational data [5]. Due to space limitations, we will only give an overview how sideways information passing and related design principles can be adapted and combined with EDN to allow scalable OLA of XML data. A detailed description can be found in the long version of this paper [10].

As in [5], we refer to all of the joins at one level of a query plan, i.e., joins that are evaluated concurrently, as a *levelwise step*. The query processing starts at the bottom level and proceeds in ascending order. By passing information among the join operations of a levelwise step $i$ an estimate $N_i$ of the final answer of the aggregation query is maintained. This estimate becomes more and more accurate as the levelwise step progresses. At any time, all available estimates $N_i$ are combined into a single estimate for the answer which will more and more converge to the exact query result.

## 3.1 Levelwise Steps

Each levelwise step is partitioned into two phases: a scan and a merge phase. The scan phase sorts the labels of all individual path solutions and finds early solutions for the whole query pattern. In the merge phase, the actual join is performed. To ensure scalability the set of path pattern solutions that are pipelined into the join operations are divided into equal-sized runs. The size of the runs is adjusted to ensure that the join can be performed in main memory.

Both phases are executed analog to the DBO System [5], but instead of tuples results of path pattern operations are processed. Figure 3 illustrates the start of the scan phase of the first levelwise step. Path pattern solutions are read into memory in a round-robin fashion by the already introduced selection operator ($i = 1$) or by the merge phase of the preceding levelwise step ($i > 1$). Subsequently, they are sorted by a hash function that only takes into account the parts of the labels that are significant for the join operation (specified by *pos*). To guarantee an unbiased label sorting the hash function is initialized with a different seed for each single binary join operation. After each sorting, all records being present in memory are immediately joined, thus, yielding early overall join results to the join conditions specified in the query pattern. Based on these early results, estimates and error bounds for the final result are generated at each step of the query execution. Due to space limitations, we omit the explanation of the estimate computation here and refer to [10].



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

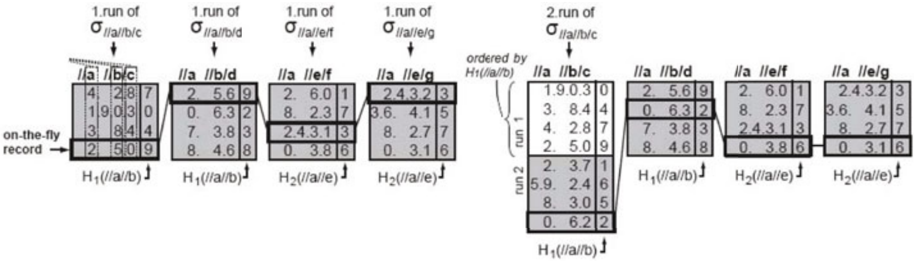


**Fig. 3.** Scan phase of the first levelwise step

The merge phase is directly connected with the scan phase of the subsequent levelwise step. It provides the following two characteristics: (i) it guarantees a (semi-)random output order and (ii) it produces output partitioned into equal-sized runs that fit into main memory. The partitioning is controlled by the values of the hash function from the scan phase. Based on equal-sized hash ranges the merge process produces runs of joined pattern in a round-robin fashion. The results of the join operations are directly streamed into the next levelwise step.

## 4 The XDBO System

We now present the architecture of the XDBO System whose main concepts were implemented in a Java prototype. As shown in the next section, this prototype is able to demonstrate the feasibility and the efficiency of our solution. However, it is designed as proof of concept, and thus, limited in its functionality (see [10] for restrictions). The XDBO System comprises two main parts which are the Import and Indexing Component (Part (I) of Figure 4) and the core XML Database System (Part II). They are supplemented by an underlying index storage.

**Import and Indexing.** The Import and Indexing Component comprises three sub-components: the Data Import Interface, the Numbering Component and a Finite State Transducer (FST) for the EDN. The Data Import Interface is used to load XML data into the system. The Numbering Component creates an element path index based on the FST that is set up for an XML document or a predefined XML schema. For each element type, this index lists the EDN labels of all nodes. Additional text indexes are created for the values of text nodes.

**Database System.** The database system follows a layered architecture with three components: a Query Execution Plan (QEP) Generator, an Execution Component and a Data Access Component. A user interface for posting queries against the system using structural XML query languages can be realized on top of XDBO. The QEP Generator accepts a structured query pattern as input and generates a QEP; it converts the given query pattern into join trees of query paths. The QEP is then handed over to the Execution Component which manages the processing of the given join tree. Joins are executed in a levelwise



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

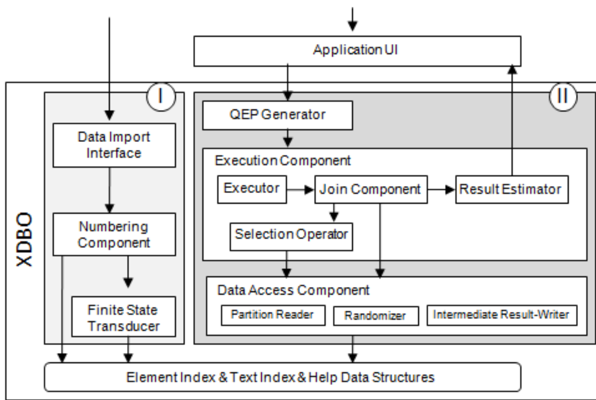


**Fig. 4.** Architecture of the XDBO System

fashion as described in Section 3.1. Additional query optimization is possible but is left for future work. On-the-fly records produced throughout the query execution are used to estimate the final aggregate with the help of the Result Estimator. All operators retrieve and store data via the Data Access Component which offers a randomization and partitioned reading functionality.

## 5 Evaluation

For the evaluation of the XDBO System, we used different sized XML documents (113 MB to 11.1 GB) generated with the XMark [9] data generator (scaling factors 1 to 100) and compared main characteristics of XDBO with a scalable implementation of the TwigStack System [2]. All experiments were conducted on a 2.4 GHz processor with 4 GB RAM running a 32-bit Windows Vista Enterprise operating system. Besides the evaluation of query processing presented as follows we had a look on the index performance, the impact of the document size and the application of the proposed optimizations. These detailed results are skipped here due to space constraints and can be found in [10].

To evaluate the query execution performance we ran several `COUNT` queries with different characteristics (selection rate, size of intermediate results, number of branching nodes) on the different sized XMark documents and picked out three of them for this paper; the respective query patterns Q1-Q3 can be found in [10]. We ran each query three times and—as each execution of a query will result in a different estimation chain which prevents averaging the results—we picked the one with the medial total execution time.

Overall, the evaluation shows that the XDBO Systems generates good estimates with confidence intervals decreasing over time. Furthermore, accurate estimates are produced long before the TwigStack System finishes execution.

**Confidence interval ratio.** First, we analyzed the relative confidence interval width defined as the ratio of the 95% confidence interval width and the query



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

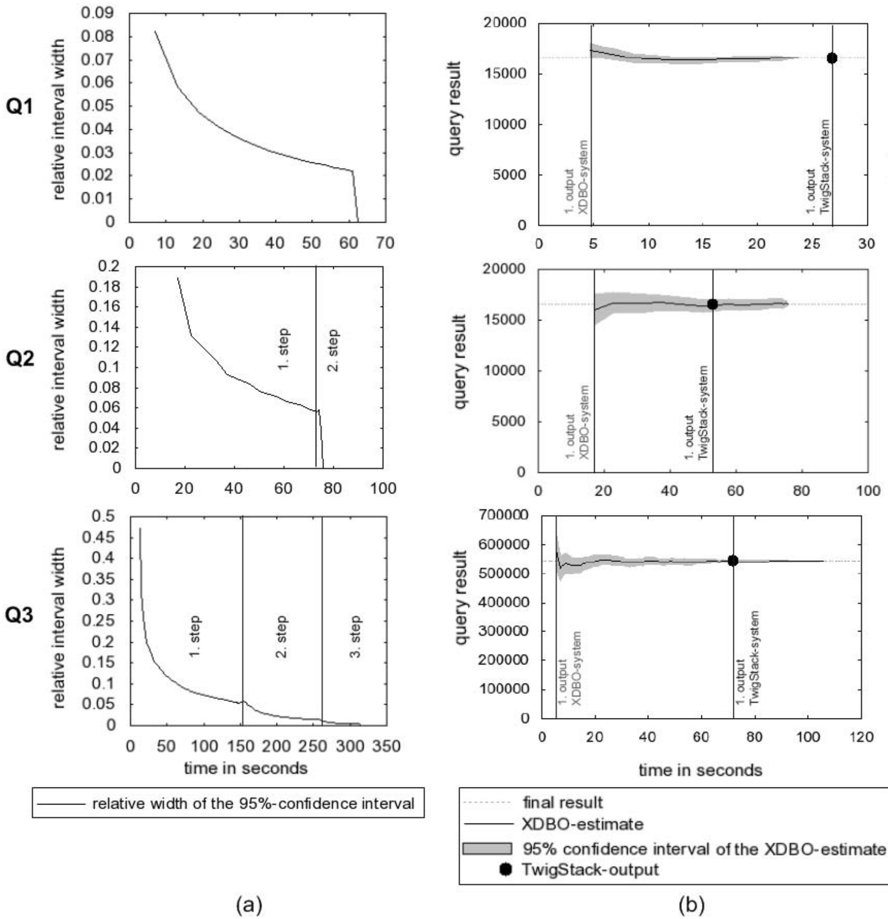


**Fig. 5.** Time evaluation of relative confidence interval width and comparison of XDBO and TwigStack System (for 2.8 GB XMark document)

estimate. Figure 5(a) shows the relative confidence interval width as a function of the processing time for a 2.8 GB XMark document (XMark scaling factor 25). A value of 0.1 implies that the 95% confidence interval equals 10% of the current estimate. The relative interval width decreases with time for all queries. The very small confidence interval of the query Q1 demonstrates very accurate estimates. In comparison, the queries with branching nodes (Q2 and Q3) show larger relative interval widths, especially at the early phase of query processing. However the interval width quickly decreases and therewith estimates are getting accurate soon. First optimization approaches to address the limitations of the relatively wide starting confidence intervals can be found in [10].

**Comparison of XDBO and TwigStack.** Figure 5(b) shows how the XDBO query execution compares with the TwigStack System for the 2.8 GB XMark



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

document. The TwigStack System returns the exact query result while XDBO outputs early estimates and error bounds as well as the exact result at the end of the query evaluation. For all queries the XDBO System was able to give first output before the TwigStack System did. For the query without branching nodes (Q1) the XDBO System not only provided fast accurate estimates but also finished before the TwigStack System was able to generate an answer. Moreover, an aggregate with a relative confidence interval width lower than 10% was produced in less than 20% of the time needed by the TwigStack System to yield the result. The queries with branching nodes required longer total execution time, but still returned accurate guesses long before the TwigStack System produced output.

## 6 Related Work

In this section, we give an overview over the different areas of related work.

**Online Aggregation.** Online Aggregation was introduced by Hellerstein et al. in [4]. A major technical challenge in OLA is to combine efficient join processing with unbiased guaranteed accuracy estimates. To address this task various algorithms such as the Ripple Join [3] or the SMS Join [6] have been proposed. Jermaine et al. [5] observed that the result inaccuracy for these algorithms significantly increases with the number of tables to be joined; their DBO System ensures scalable query processing for OLA by sharing information across relational operations at different levels of the query plan. Especially from the DBO System we adopted some concepts; however, we made significant effort to adapt these concepts to the fairly different style of query processing of XML data.

**Indexing and numbering schemes for XML data.** Indexes are a well-known technique to speed up query processing; clearly, this also holds for XML data as shown in [8]. Additionally, to speed up pattern matching a multitude of numbering techniques and algorithms have been proposed [1,7]. They encode structural relationships into labels for each element. The extended Dewey numbering scheme (EDN) [7], that we incorporate for highly efficient path pattern matching, additionally encodes the complete root-to-node paths into the labels.

**Pattern matching.** Recent algorithms for XML pattern matching exploit the characteristics of various numbering schemes to significantly improve the processing speed of structural joins. The first proposed structural join operations [1,2] focus on binary and query path patterns, but suffer from the need of additional stitching steps when applied to more complex query patterns. To process query patterns as a whole several holistic twig join algorithm had been presented [2,7]. While some of the proposed pattern matching algorithms are non-blocking they generally rely on the processing of labels in a sorted order. Like our solution, TJFast [7] exploits the features of the EDN, but does not meet the OLA requirements. Accordingly, none of these algorithms supports early result feedback and processing with guaranteed statistical error bounds.



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden

Final edited form was published in "Scientific and Statistical Database Management: 22nd International Conference. Heidelberg 2010", S.78–86, ISBN 978-3-642-13818-8
https://doi.org/10.1007/978-3-642-13818-8_7

## 7 Conclusion

In this paper, we presented the concepts and the architecture of a system capable of performing Online Aggregation over XML data. This XDBO System is able to give fast feedback to aggregation queries by approximating and refining the final answer throughout query processing. Additionally, it provides accuracy guarantees by attaching confidence information to the estimates. We introduced a novel query processing approach that splits query patterns into query path patterns; efficient query processing is realized by novel operators for selecting and joining path patterns in combination with an appropriate index structure. For accurate estimates we adapted principles of the DBO engine to the XML query processing. We prototypically implemented the XDBO System to demonstrate the efficiency of our solution. Within our extensive evaluation, we have shown that our system returns accurate guesses of the final answer long before traditional systems are able to produce output. Furthermore, good estimates are gained very fast for query patterns without branching nodes. We identified some limitations for more complex queries, but presented and demonstrated first optimization approaches to address these limitations in the long version of this paper [10].

## References

1. Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M., Srivastava, D., Wu, Y.: Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In: ICDE 2002, pp. 141–152 (2002)
2. Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: optimal XML pattern matching. In: SIGMOD 2002, pp. 310–321 (2002)
3. Haas, P.J., Hellerstein, J.M.: Ripple joins for online aggregation. ACM SIGMOD Record 28(2), 287–298 (1999)
4. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online Aggregation. In: SIGMOD 1997, pp. 171–182 (1997)
5. Jermaine, C., Arumugam, S., Pol, A., Dobra, A.: Scalable approximate query processing with the DBO engine. TODS 33(4), 1–54 (2008)
6. Jermaine, C., Dobra, A., Arumugam, S., Joshi, S., Pol, A.: A disk-based join with probabilistic guarantees. In: SIGMOD 2005, pp. 563–574 (2005)
7. Lu, J., Ling, T.W., Chan, C.-Y., Chen, T.: From region encoding to extended dewey: on efficient processing of XML twig pattern matching. In: VLDB 2005, pp. 193–204 (2005)
8. McHugh, J., Widom, J.: Query Optimization for XML. In: VLDB 1999, pp. 315–326 (1999)
9. Schmidt, A., Waas, F., Kersten, M., Carey, M.J., Manolescu, I., Busse, R.: XMark: a benchmark for XML data management. In: VLDB 2002, pp. 974–985 (2002)
10. Seidler, K., Peukert, E., Hackenbroich, G., Lehner, W.: Approximate Query Answering and Result Refinement on XML Data (Full Version). Technical report (2010), `http://wwwdb.inf.tu-dresden.de/publications`



Provided by Sächsische Landesbibliothek - Staats- und Universitätsbibliothek Dresden