

Efficient Processing of Partially Specified Twig Queries

Junfeng Zhou^{1,2}

¹School of Information, Renmin University of China, Beijing, China

²Department of Computer Science and Technology, Yanshan University, Qinhuangdao, China

zhoujf@ysu.edu.cn

ABSTRACT

XML has been used extensively in many applications as a de facto standard for information representation and exchange over the internet. Huge volumes of data are organized or exported in tree-structured form and the desired information can be got by traversing the whole tree structure using a twig query or keyword based query. Although twig query can express more semantic information, it is infeasible when the schema or structure information is not fully available. Keyword based query can be used easily; however, it can only express very limited semantics. This has driven the requirement of relaxing the **complete** specification of a twig query to express more flexible semantic information in a single query expression.

In this paper, we address the problem of query evaluation for **Partially Specified Twig Query (PSTQ)** over XML documents. We aim at (i) providing users with a concise and effective way to specify partial constraints in a twig query and (ii) processing a *PSTQ* efficiently without deriving the special twig queries and processing them one by one. The problems which prevent us from efficient processing of *PSTQ* are proposed and then, the current work for these problems is presented for further discussion.

1. INTRODUCTION

As a de facto standard for information representation and exchange over the internet, XML has been used extensively in many applications. An XML document contains hierarchically nested elements and can be naturally modeled as a tree, where nodes represent elements while edges represent direct nesting relationships between elements. Query capabilities are provided through twig queries, which are the core components for standard XML query languages, e.g. XPath[4] and XQuery[5]. A twig query can be modeled as a node labeled tree, where nodes are labeled with different tags, and edges represent either the *Parent-Child* (PC) or *Ancestor-Descendent* (AD) relationships. The two twig queries in figure 1 can be used to retrieve useful information from an XML document, where \parallel denotes ancestor-descendant

relationship and $|$ denotes parent-child relationship.

By allowing ancestor-descendant relationship, twig queries provide some freedom in the specification of a tree structure; however, they still require the precedence order between query nodes. For example, in Q_1 of figure 1(a), student must be descendant of course. This will give rise to some problems in practice.

If we don't know the schema or structure information of the given XML data set, it is impossible for us to write a query by specifying the concrete precedence relationship between nodes in a query. Even if the schema or structure information is available, when we want to get all useful information from various data sources with different structure, it will be difficult for us to use any single twig query to retrieve all useful information since any single twig query written by XPath can not express all required semantic information. For example, Q_1 can be used to retrieve useful information from XML documents shown in figure 2¹(a) and Q_2 for figure 2(b). Moreover, when the number of data sources becomes very large, it is impractical for us to submit different queries for each data source, let alone the schema or structure information is not available, repeatedly evaluating each query over all data sources is also infeasible. Traditional data integration mapping rules between a global schema and local schema are too complex and may cause errors. It is also a great burden to maintain the mapping relationship between the global structure and each data source, which may involve extensive manual intervention.



Figure 1. Two Twig Queries

To provide the user a more flexible way to specify a query when the schema or structure information of the given XML document is not available or the schema or structure of different data sources conflicts with each other, keyword based methods are proposed recently to complete the task, however, they may produce large number of useless results. Meaningful Lowest Common Ancestor, MLCA [16], can avoid some useless results

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Proceedings of SIGMOD2007 Ph.D. Workshop on Innovative Database Research 2007(IDAR2007), June 10, 2007, Beijing, China

¹ Here the element S(1) in figure 2(a) and C(1) in figure 2(b) both appears two times, which means that they refer to same object or entity in practice, we omit the reference arrow for simplicity.

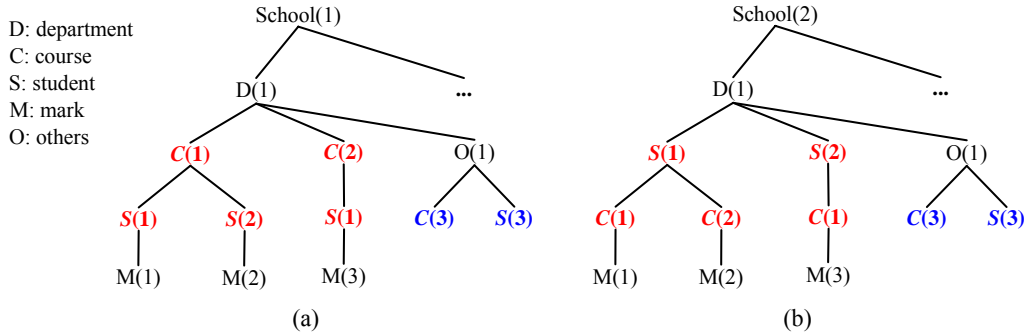


Figure 2. Two XML documents from different data sources with different structure

produced by previous methods using some semantic constraints, but it still cannot distinguish all useless results from all returned results. For example, if we use keywords “student, course” to retrieve matched information from the XML document in figure 2(a), O1 will be returned since the meaningful lowest common ancestor of C3 and S3 is O1, but actually, C3 doesn’t be taken by any student and S3 doesn’t take any course. This contradicts the meaning of C1 which is taken by S1 and S2.

In [23], the notion of Partially Specified Twig Query is proposed to tackle this problem. Compared with Twig Query, the structure information of a **Partially Specified Twig Query (PSTQ)** is partially specified, i.e. it has a tree structure, but the concrete precedence relationship between some nodes is not specified, we only know that they have ancestor-descendant relationship but not know which one is the ancestor node or vice-versa. Therefore, a *PSTQ* may derive several twig queries and we can solve structure conflict using a single *PSTQ* instead of several twig queries or keyword based query to complete the task. A naïve evaluation method for *PSTQ* can be stated as following:

Let Q be a *PSTQ*, Q_1, \dots, Q_n be twig queries corresponding to Q , R, R_1, \dots, R_n be the answers of Q, Q_1, \dots, Q_n on a database, then $R = \bigcup_{i=1}^n R_i$.

While *PSTQ* can express more flexible semantic information, it is not feasible to directly apply it to practice, because the processing cost may be too high since n may be very large using existing method [23], which will cause great burden for query processing, and some twig queries derived from the given *PSTQ* may conflict with each other considering the semantics, moreover, some derived twig queries may be actually useless, therefore, the returned results may involve large number of useless results which will cause additional processing cost for further use. Query evaluation for *PSTQ* needs us to complete the following tasks:

- **Representing a *PSTQ* in A Simple but Effective Way.** This is very important because the goal of using *PSTQ* is providing users a flexible way to express their requirement, if it is too complex and difficult for users to use, the application of *PSTQ* can not be really accepted by users.
- **Efficient Query Processing Method.** Because a *PSTQ* may correspond to several twig queries in practice, query evaluation for each twig query will result in repeatedly processing same elements. We need to design high efficient query processing methods for *PSTQs* when we do data

integration with various data sources where data is published with XML format.

- **Query Optimization for *PSTQ*.** Because a *PSTQ* may derive several twig queries in practice, it is easy for us to understand the importance of query optimization, which includes containment checking, satisfiability, minimization and view-based query evaluation. Since one *PSTQ* may correspond to several twig queries, the influence of optimization can be enlarged by all twig queries derived from the *PSTQ*.

Among them, the first problem is fundamental for user to express their requirement. The second is critical for us to improve query performance. The last problem, query optimization for *PSTQ*, is also very important for speeding up the query processing.

In this paper, we addressed the problem of query processing for *PSTQ* when (i) the structure or schema information of an XML document is not fully available or (ii) the structure or schema of different data sources conflict with each other. We aim at (i) expressing partial constraints in a twig query with a concise and effective way and (ii) processing a *PSTQ* efficiently without deriving the special twig queries and processing them one by one. We propose a general framework that enables any users (common users or expert) to query XML data exploiting whatever partial knowledge of the structure or schema information. They can, of course, write a proper XPath expression to express their requirement if they know the full schema, or they can just specify keywords if they know nothing about the structure or schema. Except that, the most important thing is that they can specify partial constraints in a twig query. Compared with keyword based query, the users can specify more semantic information in a query if they know partial (not full) structural or schema information. We also propose an algorithm which can process a *PSTQ* efficiently using a novel stack-based data structure. The difference between our method and previous stack-based methods lies in that in previous methods, the given twig query is specified with precedence relationship between query nodes, thus they can treat the query with a given order, as for our method, the concrete precedence relationship between query nodes is not specified previously, thus we don’t know the processing order of query nodes before query evaluation.

The rest of the paper proceeds as follows: Section 2 is dedicated to the description of background and related work. Section 3 discusses the current problems and challenges when processing

partially specified twig queries. Section 4 discusses my PhD work. The conclusion is presented in section 5.

2. BACKGROUND AND RELATED WORK

2.1 Data Model and Query Language

An XML document is frequently modeled as a tree in existing works, e.g. figure 2(a) and (b), where nodes represent elements while edges represent direct nesting relationships between elements. In [23], a value tree is used to represent an XML document. The two models are similar to each other when representing an XML document. When processing a twig query, we can use standard XML query language, e.g. XPath or XQuery, to express our requirements, but they both can not express the semantic information of being only at the same path.

Consider the scenario that when we extract useful data from different data sources, the integrated schema graph may involve cycles. This often happens if the objects connected by the cycle have many to many or one to one relationship in E-R graph, then different sources can organize their own data with different structure as shown in figure 2. Existing methods, e.g. twig query or keyword based query, cannot complete the task well. In [23], a new language was proposed by which we can express more flexible semantic information. The problem of this language is that it is too complex for common users.

2.2 Query Evaluation

If we want to get the matched answers for a twig query against an XML document stored in a database, we can use various query processing methods proposed in recent years to complete the task. Among them, MPMGJN[26] was proposed for efficient structural join, Stack-Tree-Desc/Anc[1] improves the query performance of MPMGJN by using stack-based binary structural join algorithm. Wu et al[25] studied the problem of binary join order selection for complex queries based on a cost model. All of these structure join methods suffer from large number of intermediate results. To process a twig query holistically, Bruno et al[6] proposed an algorithm, TwigStack, to avoid producing large size of useless intermediate results. Later works made improvements against TwigStack from various aspects. TSGeneric+[12] focused on holistic twig joins on all/partly indexed XML documents to skip some useless elements. TwigStackList [17] handles the sub-optimal problem of TwigStack when parent-child relationship is involved and it can reduce the size of intermediate results. By using extended Dewey, TJFast[18] can improve query performance by accessing only elements of leaf nodes in the query.

Further, query performance can be accelerated by filtering out large number of useless elements before query processing using structural index, e.g. DataGuide[10], 1-index[20], F&B[13] index etc. They differ in the equivalence relations, e.g. simulation and bisimulation[20,13], or even semantic equivalence relations [22]. Structural indexes have been extensively studied recently in both the exact[10,13,20,3] and the approximate flavor[14]. A common characteristic of these approaches is that the index is used as a back end for evaluating twig query without accessing the database.

By comparison, keyword based methods [9,11,16] provide us with the most flexibility. Users do not need to know the concrete document structure and the usage of query language. Lowest

Common Ancestor (LCA) can be used to identify interest segments in an XML document; however, blindly computing the LCA may cause large number of unrelated results. To meet this requirement, Meaningful Lowest Common Ancestor Structure (MLCAS) is introduced in [16] to reduce meaningless results. Although the method proposed in [16] can reduce the number of processed data elements by specifying a range for processing, the keywords in this range cannot be specified with any structural constrains.

In practice, we cannot simply assume that users know nothing about the structure or the schema of an XML document, or they fully understand these things. The notion of *PSTQ* has been proposed in [23], to the best of our knowledge, no existing work has focused on query evaluation for partially specified twig query.

2.3 Query Optimization

There is still much work focusing on optimization problems, e.g. containment, minimization and view-based query processing, to improve the query performance of twig queries.

Query containment has received more attention in the context of XML, e.g., [23, 19, 24, 21]. The satisfiability problem is a special case of the containment problem which has also been studied extensively [15]. View-based query evaluation has also been addressed in [7]. The work reported in [23] focuses on providing sound (but not necessarily complete) techniques for checking query containment that can be used for efficient processing and optimizing of *PSTQs*. However, query processing method is still not presented.

3. THE PROBLEMS AND CHALLENGES

In this section, we give out the concrete problems and point out the challengeable problems for query evaluation of *PSTQs*.

3.1 Query Language

In [23], a new query language is proposed to express the partially specified twig query. According to this language, a *Partially Specified Twig Query (PSTQ)* can be expressed as a triple $Q = (P, N, o)$, where P is a nonempty set of *Partial Paths (PPs)*. For $\forall p \in P$, the expression $e[p]$ denotes the element e in PP p . N is a set of expressions of the form $e[p_i] \equiv e[p_j]$, where p_i and p_j are PPs in P , different PPs can be connected together by these expressions; o is a PP in P which is called *output PP* of Q .

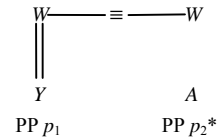


Figure 3. A partially specified twig query Q_3 [23]

Example 1: As shown in figure 3, Q_3 is a partially specified twig query, there are totally two partial paths in this query, path p_1 and path p_2^* have the common node W . In p_1 , W is an ancestor of Y , while in p_2^* , the relationship between W and A is not specified except that they are in the same path. To evaluate such a query using the method proposed in [23], we should first get the set of twig queries which can be derived from Q_3 according to dimension graph to get the relevant twig queries. The dimension

graph, as shown in figure 4(a), is another structural index proposed in [23], where the equivalence classes are formed by all the nodes labeled by the same tag name in the database., thus the processing of a *PSTQ* is done firstly on the dimension graph before being checked on the value tree, the derived twig queries is shown in figure 4(b-f). Also, these twig queries can be obtained using other structural indexes, e.g. 1-index, structural summaries, etc.

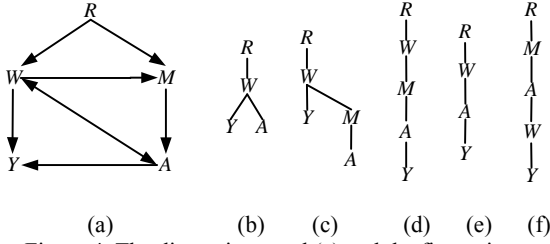


Figure 4. The dimension graph(a) and the five twig queries corresponding to Q_3 on the dimension graph (b-f) [23]

Thus we know that the query evaluation of a *PSTQ* is finally transformed into evaluating a series of twig queries derived from the given *PSTQ*. Because there is too much redundant information in a *PSTQ* when being represented by the language proposed in [23], we need to design a new language which can express the relationship of being at the same path without specifying precedence relationship. The problems about the above method can be stated as follows:

- **Too many derived twig queries.** Since we only care about three kinds of data elements, as shown in figure 4(b-f), some elements, which do not exist in the submitted query, appear in the derived twig queries, e.g. R and M. This is because only parent-child relationship is considered in the derived twig queries. In fact, the five twig queries can be expressed by two twig queries, $W[/Y]//A$ or $A//W//Y$, without losing any semantic information compared with Q_3 in figure 3.
- **Too high the processing cost.** Since there are many derived twig queries, as shown in figure 4(b-f), the processing cost may be too high, moreover, the newly added node, e.g. M and R, in the derived twig queries will cause additional processing cost.
- **May involve meaningless results.** If the users don't know the structure information of the XML document in figure 2, when they want to know that for every student, which course he has taken, they can use the query language proposed in [23] to express his requirement (figure 5(b)). The dimension graph of figure 2 is shown in figure 5(a). We can get six twig queries according to the dimension graph as shown in figure 5(c-h), we can use Q_6 and Q_7 to get all useful information from XML documents in figure 2, the worst thing is that Q_8 will cause useless information, for example, $C(3)$ and $S(3)$ in figure 2(a) satisfy Q_8 , but none of them has been taken or taken some courses. Moreover, Q_5 , Q_9 and Q_{10} will not return any results.

Based on the above observations, we need to design a new language to express the users' requirement in a simple but effective way.

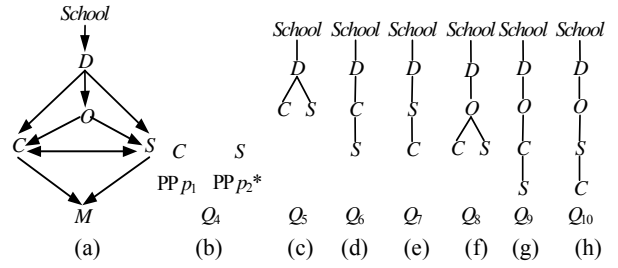


Figure 5. The dimension graph (a) of XML documents in figure 2, Q_4 (b) and the six twig queries corresponding to Q_4 on the dimension graph (c-h)

3.2 Query Evaluation

As can be seen from section 1, the query evaluation of a partially specified twig query can be done by processing all twig queries derived from the *PSTQ* one by one. Although we can process a twig query efficiently using existing structure join methods or holistic twig join methods, we need to process each element several times which equals to the number of twig queries derived from the given *PSTQ*. This will result in severe performance degradation in practice.

To the best of our knowledge, this issue has not been addressed for this type of queries. To improve the query performance of *PSTQs*, several related problems should be considered:

- (1) How to check the relationship of being at the same path between two data elements without knowing the precedence relationship?
- (2) What order should be adopted to organize the query nodes when query processing?
- (3) How to arrange the matched data elements if we use stack to compactly represent the intermediate results?
- (4) How to output the intermediate path solutions since the elements in a stack may appear at different positions in the output path solutions?

3.3 Query Optimization

Although many works have focused on optimization problems, e.g. containment, minimization and view-based query processing, to improve the query performance of twig queries, these problems become more complex for *PTPQs* since the partially specified structure allows new and non-trivial structural expressions to be derived from *PTPQs*. Unfortunately, only the work proposed in [23] has addressed containment and satisfiability problems for *partially specified twig query*.

The problem lies in the fact that from each *PSTQ* we can derive several twig queries. Existing methods can not be used directly in the context of *PSTQs*,

4. RESEARCH WORK

In this section, the problems to be solved in my PhD thesis are proposed for discussion.

4.1 Query Language

The query language proposed in [22,23] is too complex and some implied relationship in a query expression cannot be found easily since in each query expression, there are several partial paths and the precedence relationship is assigned respectively in each path. Moreover, the derived twig queries may produce meaningless results. Although in essence, a partially specified twig query has a tree structure, it is represented by several paths respectively when it is expressed by the query language in [23]. Therefore, we need to design a proper query language to simplify the representation of a $PSTQ$ without losing any semantic information.

Our method for this problem is extending the current XPath query language to support vertical relationship without specifying precedence relationship, so that we can easily express a $PSTQ$ using a single query tree in a simple way. The benefits of our method is that (1) less number of derived twig queries and (2) no meaningless twig queries.

Here we simply illustrate our method by an example.

Example 2: The query submitted by a user can be classified into three categories according to structure information: Fully specified, partially specified and no specified. For fully specified twig query, the expression is equivalent to that written with XPath. For twig query without being specified with structural relationship, like Q_4 in figure 5(b), we do not derive the corresponding twig queries as that in [23] since the derived query may cause meaningless results, we can simply express this query as $\#[//C]//S$, where $\#$ can denotes MLCA which was introduced in [16], or ICA, thus the evaluation of this query becomes very easy and the benefit is very obvious, we do not need to process many derived twig queries one by one and the meaningless results can also be avoided. For partially specified twig query, we introduce a symbol, \Downarrow , to facilitate expressing being at the same path without specifying the concrete precedence relationship. \Downarrow is used to denote the ancestor-descendant relationship or vice-versa. So query Q_3 in figure 3 can be expressed as $W[//Y] \Downarrow A$ (figure 6(a)), which is equivalent to two twig queries: $W[//Y]//A$ (figure 6(b)) and $A[//W]//Y$ (figure 6(c)). If W is dropped from p_2 of Q_3 , i.e. the common node of p_1 and p_2 disappears, when we derive the corresponding twig queries as that in [23], the processing cost will be very high, and the method proposed in [16] to process MLCA cannot tackle this kind of query since p_1 of Q_3 is a path expression but not a single node. We can express this query as $\#[//W//Y]//A$ (figure 6(d)), and here $\#$ denote MLCA or ICA according to the user's need.

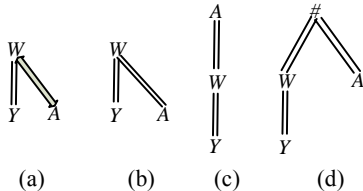


Figure 6. New form of Q_3 (a), two derived twig queries of Q_3 (b-c) and a new query(d) by dropping W from p_2 of Q_3 in figure 3.

4.2 Query Evaluation

To get the matched results for a partially specified twig query, we can do query evaluation for twig queries derived from the $PSTQ$ one by one. However, this will be too complex and the query performance may degrade severely since the number of derived twig query of a $PSTQ$ may be too large. Although existing holistic twig join methods can work efficiently, we still need to cope with the problem that a $PSTQ$ has several corresponding twig queries. To avoid producing large size of intermediate results and to avoid rescanning elements more than once, we need a new method to process query evaluation efficiently.

After carefully studying the existing holistic twig join methods, we are trying to find a new algorithm which can process a $PSTQ$ holistically and each element is processed only once. Our basic consideration is described as follows:

(1) **Vertical Relationship Checking.** This can be done very efficiently for two nodes if they have relationship of being at the same path. For two elements e_A and e_B , we just need to check whether e_A is an ancestor of e_B or e_B is an ancestor of e_A . Although the cost is two times than just checking e_A and e_B with a specific precedence relationship, it is still constant time for processing each element. Compared with the I/O cost of fetching element from disk several times, the cost can be safely ignored.

(2) **Order Arrangement for Query Node.** When doing query processing, all nodes are organized into a tree. Using the new definition Partial Solution Extension, we can process the elements corresponding to the query nodes in a query only once.

Here we use C_q to denote the element pointed by C_q in T_q , to expressing convenience, we use " C_q " or "element C_q " denotes the element pointed by C_q when there is no ambiguity. C_q can point to the element next to current one by using $Advance(C_q)$, at the same time, we can use $C_q.start$, $C_q.end$ and $C_q.level$ to get the attribute values of C_q .

Definition1: Partial Solution Extension (PSE). We say a query node q has a PSE iff q satisfies any one of the following conditions:

- (1) If q is a leaf node, C_q does not equal to NULL.
- (2) If q is a non-leaf node, for each $q' \in children(q)$
 - (i) If $q // q'$, then C_q is ancestor of $C_{q'}$.
 - (ii) If $q \Downarrow q'$ (same path relationship) and q' has a PSE, C_q can cover $C_{q'}$ or be covered by $C_{q'}$ or $C_q.end < C_{q'}.start$.
 - (iii) If $q \Downarrow q'$ and q' hasn't PSE, let p be descendent of q' which has PSE, then $C_q.start < C_p.start$.

Lemma 1: For a $PSTQ$ Q , if query node $q \in Q$ has a PSE, then C_q must correspond to a solution extension[12] of a twig query which can be derived from Q , thus it may participates in a final solution.

We designed a function getNext to be called repeatedly to get an element for processing, if this element can participate in the final results, it will be pushed into a stack. Because a query node in the query may not denote the actual position of an element in the final results, which query node should be returned is very important especially when there is nesting document structure.

(3) **Stack Organization.** To compactly represent the temporal results, we need a stack for each query node, and we have designed a new strategy to organize the elements, the results will be organized in terms of the twig queries derived from a *PSTQ*. The main problem here is how to push an element into stack since an element may participate in results of different derived twig queries.

(4) **Result Output.** The method we use to output the results is similar to that of TwigStack. The main consideration here is that path solutions corresponding to different derived twig queries are outputted according to the path in the *PSTQ* and the final results are outputted in the correct order in terms of different twig queries.

As a valuable problem, we will focus on the work to use structural index to accelerate the query processing by filtering out large number of useless data elements.

Although optimization, e.g. containment, minimization and view-based query processing, have been studied extensively in recent years for efficient evaluation of XPath expression in the presence and the absence of schemas, only the work in [22, 23] has addressed the containment and satisfiability problem for *PSTQ*. We will continuously keep our attention to this problem.

5. CONCLUSIONS

Querying capabilities are provided through queries with branching path expressions since there are huge volumes of data which are organized or exported in a tree-structured form. However, existing methods for getting useful information from tree-structured document are not effective enough. In this paper, we discuss the method for semantically querying tree-structured data sources using partially specified twig queries which can be used to query multiple trees with structural differences. After summarizing the current research status in this area, we introduce the issues which are being focused on now and will be addressed in the future.

6. REFERENCES

- [1] S. Al-Khalifa, H. V. Jagadish etc. Structural Joins: A primitive for efficient XML query pattern matching. In *Proceedings of ICDE 2002*, pages 141-152, 2002.
- [2] S. Amer-Yahia, S. Cho etc. Tree Pattern Relaxation. In *Proceedings of the 8th International Conference on Extending Database Technology*, 2002.
- [3] A. Barta, M. P. Consens, A. O. Mendelzon. Benefits of Path Summaries in an XML Query Optimizer Supporting Multiple Access Methods. In *Proceedings of VLDB 2005*, pages 133-144, 2005.
- [4] A. Berglund, S. Boag etc. Simeon. XML Path Language (XPath) 2.0. W3C Working Draft 22 August 2003.
- [5] S. Boag, D. Chamberlin etc. XQuery 1.0: An XML Query. W3C Working Draft 22 August 2003.
- [6] N. Bruno, N. Koudas, D. Srivastava. Holistic twig joins: Optimal XML pattern matching. In *Proceedings of SIGMOD 2002*, pages 310-321, 2002.
- [7] D. Calvanese, G. D. Giacomo etc. What is view-based query rewriting (position paper). In *Proceedings of KRDB*, pages 17-27, 2002.
- [8] S. Chen, H.G. Li etc. Twig²Stack: Bottom-up Processing of Generalized-Tree-Pattern Queries over XML Documents. In *Proceedings of VLDB 2006*, pages 283-294, 2006.
- [9] S. Cohen, J. Mamou etc. XSearch: A Semantic Search Engine for XML. In *Proceedings of VLDB*, 2003.
- [10] R. Goldman, J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of VLDB 97*, pages 436-445, 1997.
- [11] V. Hristidis, Y. Papakonstantinou etc. Keyword Proximity Search on XML Graphs. In *Proceedings of ICDE2003*, pages 367-378, 2003.
- [12] H. Jiang, W. Wang etc. Holistic twig joins on indexed XML documents. In *Proceedings of VLDB 2003*, pages 273-284, 2003.
- [13] R. Kaushik, P. Bohannon etc. Covering indexes for branching path queries. In *Proceedings of SIGMOD 2002*.
- [14] R. Kaushik, P. Sheony etc. Exploiting local similarity for efficient indexing of paths in graph structured data. In *Proceedings of ICDE 2002*, pages 129-140, 2002.
- [15] L. V. S. Lakshmanan, G. Ramesh etc. On Testing Satisfiability of Tree Pattern Queries. In *Proc of VLDB2004*, pages 120-130, 2004.
- [16] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *Proceedings of VLDB2004*, pages 72-83, 2003.
- [17] J. Lu, T. Chen etc. Efficient Processing of XML Twig Patterns with Parent Child Edges: A Look-ahead Approach. In *Proceedings of CIKM 2004*, pages 533-542, 2004.
- [18] J. Lu, T.W. Ling etc. From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching. In *Proceedings of VLDB*, pages 193-204, 2005.
- [19] G. Miklau, D. Suciu. Containment and equivalence of a fragment of XPath. *Journal of the ACM*, vol. 51, no. 1, pages 2-45, 2004.
- [20] T. Milo, D. Suciu. Index structures for path expressions. In *Proceedings of ICDT 99*, pages 277-295, 1999.
- [21] F. Neven, T. Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proceedings of ICDT2003*, pages 315-329, 2003.
- [22] D. Theodoratos, T. Dalamagas etc. Semantic Querying of Tree-Structured Data Sources Using Partially Specified Tree Patterns. In *Proceedings of CIKM*, pages 712-719, 2005.
- [23] D. Theodoratos, S. Souldatos etc. Heuristic Containment Check of Partial Tree-Pattern Queries in the Presence of Index Graphs. In *Proceedings of CIKM*, pages 445-454, 2006.
- [24] P. T. Wood. Containment for XPath fragments under DTD constraints. In *Proceedings of ICDT*, pages 300-314, 2003.
- [25] Y. Wu, J. M. Patel etc. Structural Join Order Selection for XML Query Optimization. In *Proceedings of ICDE 2003*, pages 443-454, 2003.
- C. Zhang, J.F. Naughton etc. On Supporting containment Queries in Relational Database Management Systems. In *Proceedings of SIGMOD2001*, pages 425-436, 2001.