Optimize Twig Query Pattern Based on XML Schema

Hui Li
Beijing University of Technology, Beijing, China
Email: xiaodadaxiao2000@163.com

HuSheng Liao and Hang Su
Beijing University of Technology, Beijing, China
Email: {liaohs, suhang}@bjut.edu.cn

***Abstract*—Aiming at the core operation for XML data queries—Tree Pattern Queries, we propose an effective method for minimizing XML query based on XML Schema. The method can make use of the structure constraints information provided by XML Schema to optimize the tree query pattern containing the logic nodes AND and OR. Such tree pattern preferably satisfies requirements of XQuery queries describing and optimizing to support the high-performance implementation of XQuery.**

***Index Terms*—XML, XQuery, XML Tree Pattern Queries, XML Schema**

I. Introduction

Query pattern composed of several structural joins often appears in the query request described by XML query languages such as XPath and XQuery. This kind of tree pattern queries is called Twig Query, also called Tree Pattern Query (TPQ), which is considered as the core operation for XML data queries, while the extensible markup language XML has become indispensable in many areas[1]. However, TPQ often contains redundant sub-queries, which will bring additional expenses of the query processing. People have developed a variety of methods of minimizing TPQ. With the development of XML processing technology, more features in XQuery queries are included in TPQ semantics by adding logical nodes and weak bindings. Nevertheless, the existing XML query minimization techniques have not yet to support the new features. Therefore, the paper makes a study of minimization of expanded TPQ based on XML Schema. The main contributions of the paper are as follows:

(1) Aiming at the TPQ with logical nodes and weak bindings, we propose an effective method of minimizing TPQ based on XML Schema. The method can remove the unnecessary query nodes and logical nodes according to the Required Parent-Child Constrains (RPC) and Required Ancestor-Descendant Constraints (RAD) between the XML nodes extracted from given XML Schema.

(2) We develop and realize PRC and RAD extraction algorithm from XML schema without recursive defined element type.

(3) We design and implement a TPQ minimization algorithm with O(n) time and space. The effectiveness of the algorithm is demonstrated by testing.

In the following sections, Section 2 introduces the related works and Session 3 describes the motivation of the work. Session 4 and Session 5 describes the algorithm of extracting RPC and RAD from XML Schema and algorithm of minimizing Twig queries respectively. Session 6 gives the experiment and the analysis of the result and Session 7 gives the conclusion.

II. Related work

There is a variety of methods of optimizing query of XML data , such as references [2] and [3]. Reference [3] makes a systematic and comprehensive study of minimization of XML query. The methods of minimizing TPQ can be divided into two categories, one for the query patterns which have redundancy themselves, another for Twig query patterns which are redundant under the constraints extracted from the given XML schema. Amer-Yahia[4] presented an $O(n^4)$ algorithm for minimizing TPQ in the absence of constraints and presented an $O(n^6)$ algorithm for minimizing TPQ in the presence of RPC and RAD. Ramanan[5] presented $O(n^2)$ algorithm of minimizing TPQ in the absence of constraints, $O(n^4)$ algorithm in the presence of RPC, RAD and subtype, and $O(n^2)$ algorithms in the presence of RPC and RAD.

In order to enrich query semantics of TPQ, reference [6] presents the concept of GTP, extending TPQ with the weak binding. Reference [7] presents the concept of ATP (Annotated Pattern Tree), adding predicate information on the basis of GTP. References [8], [9] and [10] add logical nodes and wildcard node test to TPQ respectively. However, the minimization methods above do not support the minimization of the TPQ which contains logical nodes and weak bindings. We propose a method of minimizing TPQ to support the optimization of extended GTPs that contain weak bindings, logical nodes AND and OR.

Manuscript received January 31, 2013; revised February 22, 2013; accepted February 28, 2013.

Copyright credit, project number: 4122011 and 61202074. Corresponding author: Hui Li.

III. Motivation

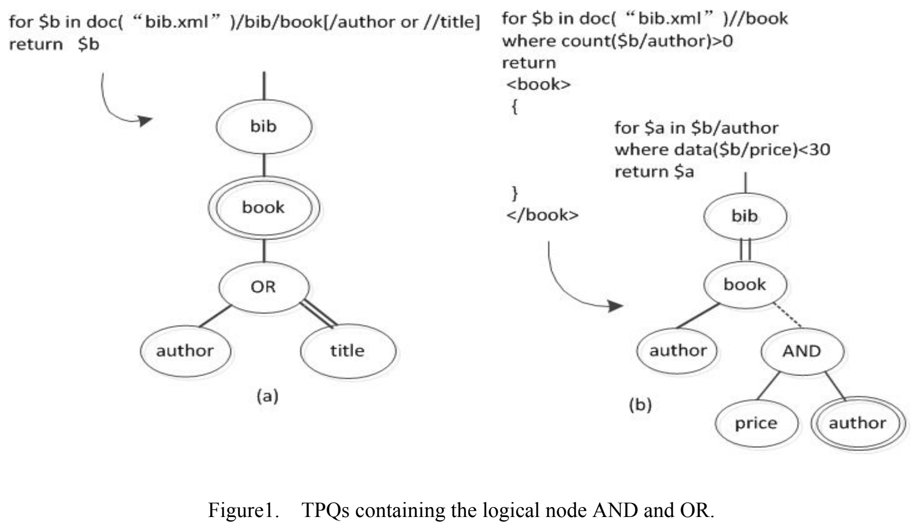
The standard TPQ contains structural joins of Parent-Child relations (PC) and Ancestor-Descendant relations (AD) only, and this kind of query pattern exists in XPath expressions and FLOWR expressions of the XQuery. However, there may be correlations among multiple TPQs in nested FLWOR expressions and different modules of XQuery. Several relevant TPQs can be combined into an extended Twig query, i.e. GTP query, by adding weak bindings and logical nodes, so that multiple TPQs can be performed by a single GTP query. For example, Fig. 1, shows two TPQs extracted from two XQuery programs. In Fig.1(a), the query requests the *book* nodes which must be the child of *bib* and have either *author* node or *title* node as its child, and TPQ in Fig.1(a), expresses the query requirements. The nodes with XML label in TPQ are called *query nodes***,** to match XML nodes in the XML documents. Each edge in TPQ represents a structural constraint between two XML nodes. The single-line edge represents PC (Parent-Child) relationship and the double-line edge represents AD (Ancestor-Descendant) relationship. In TPQ shown in Fig.1(a), the double-line edge which connects the *title* node and logical node OR indicates an AD relationship between them. The double-circle node indicates a return node of TPQ. For the query in Fig.1(b), a structural constraint is given in *for* clause and *where* clause. That is, a *book* node must have a child node labeled with *author*. Therefore, the structural constraint in TPQ represented as a strong binding. There may be queries in the return clause which is applied on the results of the previous queries. These results are dispensable and the feature can be supported by weak binding in TPQ. In GTP queries, the solid edge shows the strong binding relationship between nodes while the dotted edge indicates the weak binding relationship. Moreover, the results of upper queries are used in *for* clause and *where* clause in the internal FLOWR expressions, and the *author* node and *price* node must exist simultaneously. However, the result of the internal FLWOR expression is dispensable. Therefore, we extend GTP with the logical node AND to connect two strong bindings for expressing such special requirement, while a weak binding is used for the logical node AND. As shown in the Fig.1(b), the *price* and *author* nodes are connected with a strong binding and the node AND itself is connected with a weak binding.

There are still opportunities for optimization in GTP queries which contain logical nodes and weak bindings. For example, the structural constraint information, which can be identified from XML Schema shown in Fig.2, includes RPC (*book*, *title*) and RAD (*book*, *name*). That is, every *book* node must have a *title* node as its child and a *name* node as its descendant. Accordingly, TPQ in Fig.3(a) can be optimized as follows. Because the semantics of the logical node OR only requires one branch's condition to be satisfied, the constraint RAD(*book*, *name*) indicates the condition of the branch *//name* has been satisfied. Therefore, the logical node OR can be removed and the optimized TPQ is shown on the right side of Fig.3(a). For another example, the TPQ shown in Fig.3(b), has a logical node AND. If it is able to get the structural constraints RPC(*book*, *title*), RPC(*title, @year*) and RAD(*book*, *name*), the two branches' conditions of the logical node AND are satisfied and the logical node AND can be removed and the optimized TPQ is shown on the right in Fig.3(b). Although the structural constraints RAD (*book*, *name*) can be extracted from XML Schema and the requirement of the logical node OR has been satisfied, the *title* node cannot be removed because it is the return node. However, as shown in Fig.3(c), the logical node OR can be removed at least.

These cases illustrate GTPs with logical nodes can be also optimized based on the structural constraints provided by XML Schema. Therefore, this paper studies an effective method for minimizing queries to implement the optimization of this kind of TPQ queries.

IV. The algorithm of extracting RPC and RAD relationships

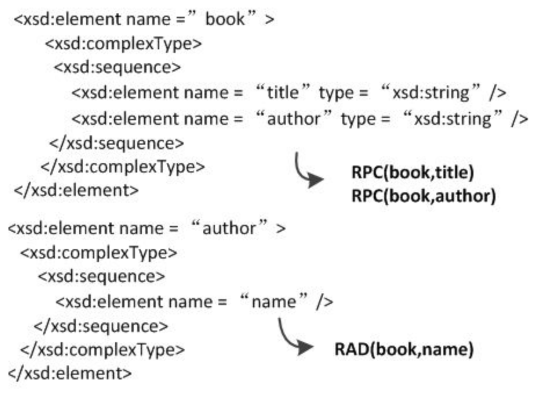
XML Schema information can be used to judge whether the relationship between two XML nodes can meet RPC or RAD relationship. In order to check the



```
for $b in doc( "bib.xml" )/bib/book[/author or //title]
return  $b
```

```
for $b in doc( "bib.xml" )//book
where count($b/author)>0
return
 <book>
 {
          for $a in $b/author
          where data($b/price)<30
          return $a
 }
</book>
```

Figure1. TPQs containing the logical node AND and OR.



```
<xsd:element name =” book” >
    <xsd:complexType>
     <xsd:sequence>
        <xsd:element name = “title” type = “xsd:string” />
        <xsd:element name = “author” type = “xsd:string” />
     </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name = “author” >
  <xsd:complexType>
     <xsd:sequence>
        <xsd:element name = “name” />
     </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

RPC(book,title)
RPC(book,author)
RAD(book,name)

Figure 2. XML Schema and the corresponding RPC and RAD relationships.

RPC or RAD relationships efficiently, as the pretreatment of TPQ minimization, these algorithms getAllRChild and getAllRDescendant are used to obtain the required children collection and the required descendant collection of each node in XML Schema respectively, which are the node collections satisfying PRC or RAD relationships with the current node. The specific algorithms are described as follows

Algorithm *getAllRChild(xsd)*
Input: *xsd*: XML Schema document
Output: the required children collection of each element in XML Schema

```
1. Begin
2.   for each global element in xsd do
3.      insert the element into the queue handleElement;
4.      for each element in the handleElement do
5.         add its child element whose number of its occurrences is
           greater than zero into the required children collection of
           the element ;
6.         add the attributes of the element marked as ‘required’
           into the required children collection of the element;
7.         if any child of the element is not in the handleElement
           then
8.            add the child into the queue handleElement;
9.   End
```

Algorithm *getAllRDescendant(xsd, sets)*
Input: *xsd*: XML Schema document
sets: each element and its required children collection in XML Schema
Output: the required descendant collection of each element in XML Schema

```
1.  Begin
2.    for each element without child and descendant in xsd do
3.       insert the element into the queue handleElement;
4.    for each element in the handleElement do
5.       let pa be parent of the element
6.       merge the required children collection of the element in
         sets into the required descendant collection of pa;
7.       merge the required descendant collection of the element
         into the required descendant collection of pa;
8.       merge the required children collection of pa into the re-
         quired descendant collection of pa;
9.       if every child of pa is in handleElement then
10.          insert pa into handleElement;
11. End
```

The algorithm *getAllRchild* above is used to get all RPC relationships and the algorithm *getAllRDescendant* is used to get all RAD relationships. The two algorithms use a queue *handleElement* to maintain the elements to be processed respectively. The former uses a top-down method and get the required children of each element gradually. The latter uses a bottom-up approach and calculate the required descendant of each element gradually. Here, we assume that there is no recursive defined element type in XML Schema, which can be represented by a directed acyclic graph.

V. Optimization of Twig query pattern

In order to describe the algorithms easily, we give several definitions.

Definition 5.1 Given an XML Schema, for any PC edge *(x,y)* in TPQ, if *x* marked as *a*, *y* marked as *b* and there is a RPC(*a*, *b*) relationship in this XML Schema, then *x* and *y* are said to satisfy the RPC relationship, denoted as RChild(*y*).

Definition 5.2 Given an XML Schema, for any AD edge *(x,y)* in TPQ, if *x* marked as *a*, *y* marked as *b* and there is a RAD(*a*, *b*) relationship in this XML Schema, then *x* and *y* are said to satisfy the RAD relationship, denoted as RDesc(*y*).

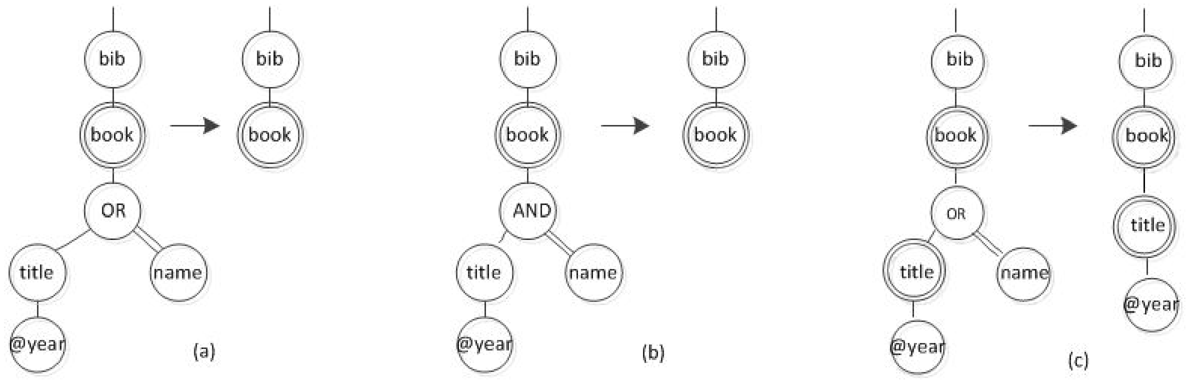


Figure 3. The examples of optimization of GTP.

TABLE I.
THE OPTIMIZATION RULES OF QUERY NODE X

	PC/AD	binding	Red(x)	Ret(x)	RChild/RDesc	processing method
1	PC	any	true	false	RChild	remove x's sub-tree
2	AD	any	true	false	RDesc	remove x's sub-tree
3	PC	weak	true	true	RChild	become a strong binding
4	AD	weak	true	true	RDesc	Become a strong binding

Definition 5.3 Given an XML Schema, for any query node *x* in TPQ, if every PC edge meets the RPC relationship and every AD edge meets the RAD relationship in the sub-tree rooted at *x*, then the sub-tree is said to be redundant, denoted as Red(*x*).

Definition 5.4 For any node *x* in a TPQ, if there is at least one return node in the sub-tree rooted at *x*, then the sub-tree must be retained, denoted as Ret(x).

Definition 5.5 For any logical node *x* in TPQs, let *y* be an ancestors of *x*, if *y* is the nearest query node from *x*, then *y* is called real parent of *x*, denoted as realparent(*x*). As shown in Fig.3(a), the real parent of OR node is the *book* node, because it is a nearest query node from the OR node in its ancestors.

A. Optimization of Query Node

The nodes in TPQs are divided into two kinds. One is the query node, which expresses the requirements of the XML node to match. Another is the logical node, which represents the logical relationship between the various matching requirements. The algorithm of minimizing TPQ in the paper carries on TPQ optimization from bottom to top. If the current node *x* and its parent are both query nodes, then optimize *x* according to the rules in Table I. If the current node is a query node and its parent node is a logical node, then we need to calculate Red(x) and Ret(x). If the current node is a logical node, we optimize the logical node according to the optimization algorithm of OR and AND respectively. After all children of the parent of the current node have been optimized, the parent node should be optimized. So we carry on a traversal with all the nodes in the process of optimizing. The specific algorithm *TreeOptimize* is as follows:

```
Algorithm  TreeOptimize(root)
Input:         root :  the root of tree pattern
Output:        the optimized tree pattern

1.  Begin
2.  put all leaf nodes of tree pattern into the queue handleNode ;
3.  for  each node x in the handleNode  do
4.      if  x and its parent are query nodes  then
5.          if  x and its parent meet RPC/RAD  then
6.              if  Ret(x)=false  then
7.                  if  Red(x)=true  then
8.                      remove the x 's sub-tree from tree pattern;
9.              else if  Ret(x)=true  then
10.                 if  Red(x)=true and x connects its parent by
                    a weak binding  then
11.                     change it into a strong binding;
12.     else if  x is logical node  then
13.         if  x is OR  then
14.             optimize it by OrOptimize(x);
15.         else if  x is AND  then
16.             optimize it by AndOptimize(x);
17.     if  every child and descendent of the parent of x are in
        handleNode  then
18.         put the parent of x into handleNode;
19. End
```

B. Optimization of the Logical Node OR

The logical node OR may have multiple sub-trees, the RPC and RAD constraints obtained from XML Schema can make some sub-tree's requirement of OR satisfied, and then make the requirement of OR satisfied. As shown in Fig.3(a), the *book* node has two sub-trees */title/@year* and *//name*. If we get RAD (*book*, *name*), or RPC(*book*, *title*) and RPC(*title*, *@year*) constraints simultaneously, the requirement of OR is met and can be optimized. The optimization rules are given in Table II.

The basic principles are: (1) If the requirement of OR is met, we should remove the logical node OR. (2) For each child node *y* of an OR node, if *y* and its real parent satisfy RPC/RAD, Red(y) is true and there is no return node in the sub-tree, then the sub-tree rooted at *y* can be removed. (3) If *y* and its real parent satisfy RPC/RAD, Red(y) is true and there is at least one return node in the sub-tree, then we create a structural connection between *y* and its real parent by a strong binding. (4) If Red(y) is false or *y* and its real parent do not satisfy RPC/RAD, and there is at least one return node in the sub-tree, then we create a structural connection between *y* and its real parent by a weak binding. (5) If Red(y) is false or *y* and its parent do not satisfy RPC/RAD, and there is no return node in the sub-tree, then the sub-tree can be removed.

TABLE II.

THE SUB-NODE Y OF OR AND THE OPTIMIZATION RULES OF ITS SUB-TREE

	PC/AD relationship	The requirement of OR is satisfied	Red(y) and RChild	Red(y) and RDesc	Ret(y)	Processing method
1	PC	true	true	any	true	remove OR, add y's sub-tree into realparent(OR) by a strong binding
2	AD	true	any	true	true	ibid.
3	PC	true	true	any	false	remove OR and y 's sub-tree
4	AD	true	any	true	false	ibid.
5	PC	true	false	any	true	add y's sub-tree into realparent(OR) by a weak binding
6	AD	true	any	false	true	ibid.
7	PC	true	false	any	false	remove y's sub-tree
8	AD	true	any	false	false	ibid.

For example of TPQ shown in Fig.2(a), if it is able to obtain constraints *PRC(book, title)* and RPC*(title, @year)*, then requirement of OR is met and OR can be removed. The two sub-trees of OR can also be removed because there is no return node in them. For another example in Fig.4(a), there are return nodes in sub-tree. If constraints *RPC(book, title)* and *RPC(title, @year)* can be obtained, then the requirement of the sub-tree */title/@year* is satisfied. So the OR's requirement is met and OR can be removed. At this time, because there is a return node labeled with *title*, this sub-tree should be added into its real parent by a strong binding. While the requirement of the sub-tree *//name* is not met and there is a return node labeled with *name* in this sub-tree, this sub-tree should be added into its real parent by a weak binding.

The algorithm of minimizing the logical node OR is as follows. The method is to optimize every sub-tree in turn. Because the optimization algorithm is carried out from bottom to top, so when we optimize the logical node OR, each sub-tree of OR has been optimized. During processing, it is necessary to record whether the requirement of OR has been satisfied, and then judge it according to the rules in Table II.

Algorithm *OrOptimize(x)*
Input: the logical node *x*
Output: the optimized logical node *x*

```
1. Begin
2.  for each sub-tree of x do
3.    let y be the root of the sub-tree
4.    if the sub-trees before y 's sub-tree do not meet the requirement
        of x then
5.      if Ret(y)=true and Red(y)=true and RChild(y) or RDesc(y)
        then
6.        remove x;
7.        add y's sub-tree into realparent(x) by a strong binding;
8.        if the parent of x is a logical node OR then
9.          mark its parent's requirement as met;
10.       for each sub-tree before y's sub-tree do
11.         let c be the root of the sub-tree
12.         if Ret(c)=true then
13.           add c's sub-tree into realparent(x) by a weak
              binding;
14.         else
15.           remove c's sub-tree;
16.     else if Ret(y)=false and Red(y)=true and RChild(y) or
        RDesc(y) then
17.       remove y's sub-tree and x;
18.       if the parent of x is a logical node OR then
19.         mark its parent's requirement as met;
20.       for each sub-tree before y's sub-tree do
21.         let c be the root of the sub-tree
```

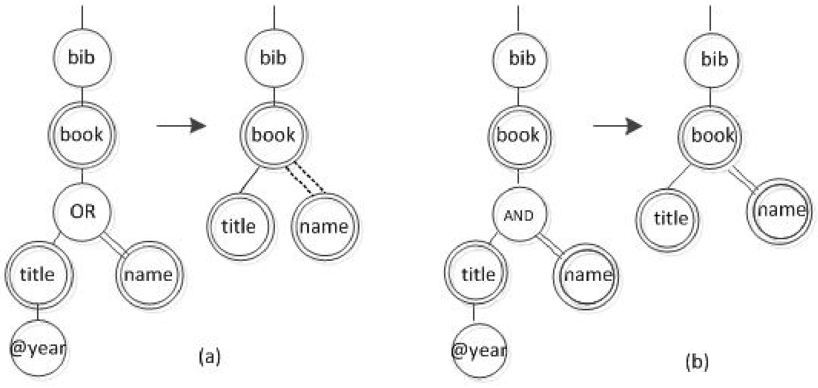


Figure 4. The examples of optimization of sub-tree of logical node containing return nodes

```
22.             if  Ret(c)=true  then
23.               add c 's sub-tree into realparent(x) by a weak
                  binding;
24.             else
25.               remove c's sub-tree;
26.     else if the sub-trees before y's sub-tree make the requirement of x
        satisfied  then
27.         if  x is still in tree pattern  then
28.             remove x from tree pattern;
29.         if  the parent of x is a logical node OR  then
30.             mark its parent's requirement as met;
31.         if  y is a logical node and Ret(y)=true  then
32.             add y's sub-tree into realparent(x) by a weak binding;
33.         else if Ret(y)=true and Red(y)=true and RChild(y) or
            RDesc(y)  then
34.               add y's sub-tree into realparent(x) by a strong bind-
                  ing;
35.          else if Ret(y)=true and (Red(y)=false or RChild(y)=false
               or RDesc(y)=false)  then
36.               add y's sub-tree into realparent(y) by a weak bind-
                  ing;
37.          else
38.               remove y's sub-tree;
39.     End
```

C. Optimization of the Logical Node AND

The logical node AND has multiple branches in TPQ. According to the semantics of AND, the requirement of node AND is met only when the requirements of all the sub-trees of AND are met. As shown in Fig.3(b), the logical node AND has two sub-trees, */title/@year* and *//name*. If it is able to get constraints *RPC(book, title)*, *RPC(title, @year)* and *RAD(book, name)*, then the requirement of the logical node AND is met and the node AND can be moved. Table III gives the optimization rules for the child node *y* of AND and its sub-tree. The basic principles are: (1) Only consider the situation that *Red(y)* is true and *y* and its real parent satisfy RPC/RAD. (2) If we can confirm the satisfaction of the requirement of AND, then remove the logical node AND. (3) If there is no return node in the sub-tree, this sub-tree can be removed. (4)If there is at least one return node in the sub-tree, add the sub-tree into its real parent by a strong binding.

For example of TPQ as shown in Fig.3(b), if we can get the constraints RPC(*book*, *title*), RPC(*title, @year*) and RAD(*book*, *name*), then the requirement of node AND is met and it can be removed. Because the two sub-trees do not have a return node, so the two sub-trees can also be removed. For another example, TPQ shown in Fig.4(b), if we can get the constraints *RPC(book, title), RPC(title, @year)* and *RAD(book, title),* the requirement of AND is met and the logical node AND can be removed from tree pattern. Because the two sub-trees both have return nodes and the requirements of the two sub-trees have been satisfied, so we add the two sub-trees into its real parent by a strong binding, as shown on the right in Fig.4(b). The algorithm is as follows.

```
lgorithm    AndOptimize(x)
Input:   the logical node x
Output:  the optimized logical node x

1.   Begin
2.       i = 0;
3.       for  each sub-tree of x    do
4.           let y be the root of the sub-tree
5.           if Red(y)=true and RChild(y) or RDesc(y)   then
6.               i++;
7.               if  i = the number of x's branches    then
8.                   remove x;
9.               if  Ret(y) = true   then
10.                  add y's sub-tree to realparent(x) by a strong
                     binding,;
11.              else
12.                  remove y's sub-tree;
13.  End
```

D. Computing Complexity

The query minimization approach includes the algorithms *TreeOptimize, OrOptimize*, *AndOptimize*. In *TreeOptimize*, TPQ is traversed completely once only. Therefore it is a linear time algorithm. Algorithms *OrOptimize* and *AndOptimize* are applied only on every logical node OR and And respectively and logical nodes is much less than query nodes typically. But they are based on

TABLE III.
THE SUB-NODE Y OF AND AND THE OPTIMIZATION RULES OF ITS SUB-TREE

	PC/AD relationship	The requirement of AND has been satisfied	Red(y) and RChild	Red(y) and RDesc	Red(y)	Processing method
1	PC	true	true	any	true	remove AND and add y's sub-tree to realparent(AND) by a strong binding
2	AD	true	any	true	true	ibid.
3	PC	false	true	any	true	add y's sub-tree to realparent(AND) by a strong binding
4	AD	false	any	true	true	ibid.
5	PC	true	true	any	false	remove AND and y's sub-tree
6	AD	true	any	true	false	ibid.
7	PC	false	true	any	false	remove y's sub-tree
8	AD	false	any	true	false	ibid.

TABLE IV.
THE TEST DATA

	the number of elements in XML Schema	The number of return nodes in TPQ	the number of logical nodes		the number of TPQ nodes before optimization	the number of TPQ nodes after optimization
			AND	OR		
1	20	1	0	0	4	2
2	20	3	0	0	4	3
3	20	2	2	0	8	4
4	20	2	0	2	8	7
5	20	3	2	2	16	10
6	40	1	0	0	4	3
7	40	3	0	0	4	3
8	40	1	2	0	8	6
9	40	1	0	2	8	2
10	40	3	2	2	16	14
11	60	1	0	0	4	1
12	60	2	0	0	4	2
13	60	1	2	0	8	7
14	60	1	0	2	8	4
15	60	3	2	2	16	8
16	80	1	0	0	4	1
17	80	2	0	0	4	2
18	80	1	2	0	8	7
19	80	1	0	2	8	6
20	80	3	2	2	16	9

Red(x) for every node *x* in TPQ. The computation of *Red(x)* can be also completed in the time and space *O(n)* where *n* is the number of TPQ nodes.

VI. EXPERIMENT

In the test section, we test in different cases of the number of XML Schema's elements, the number of return nodes in TPQ and the number of the logical nodes in TPQ. We get a conclusion by comparing the number of nodes in TPQ before and after optimization. Detailed test data are shown in Table IV.

Through the experiments above, we know that the more PRC/RAD relationships are obtained from XML Schema, the more nodes in TPQ can be reduced. For the optimization of logical node, if there are more logical nodes OR in Twig pattern, the number of nodes in Twig pattern can be reduced greatly, because the logical node OR only requires one of its sub-tree's requirement to be satisfied. While the logical node AND can be optimized only when all its sub-tree's requirements are met. Meanwhile, the number of return nodes in TPQ can also affect the optimization. Since the return nodes are not redundant, they cannot be removed in any case. If there are many return nodes in TPQ, no much optimization can be realized.

VII. CONCLUSION

This paper proposes an effective method for minimizing XML query based on XML Schema for extended Tree Pattern Queries. The method can make use of the structure constraints information provided by XML Schema to optimize the tree query pattern containing the logic nodes AND and OR. We have developed and realized PRC and RAD extraction algorithm and TPQ minimization algorithm. The effectiveness of the algorithm is demonstrated by testing.

The future works is to identify more constraints from XML schema and apply them to optimize various extended GTPs. The XML schema with the recursive defined element types should be also taken into account.

ACKNOWLEDGMENT

The author wishes to thank HuSheng Liao and Hang Su for their patient guidance. This work was both supported in part by the Beijing Nature Science Foundation under Grant 4122011and the National Science Foundation for Young Scientists of China under Grant 61202074.

REFERENCESE

[1] Christopher League and Kenjone Eng, Schema-Based Compression of XML Data with Relax NG, Journal of Computers, Vol 2, No 10(2007), 9-17, Dec 2007, doi: 10.4304/jcp.2.10.9-17.

[2] Fatma Zohra Bessai-Mechmache and Zaia Alimazighi, Aggregated Search in XML Documents, Journal of Emerging Technologies in Web Intelligence, Vol 4, No 2(2012), 181-188, May 2012 doi: 10.4304/jetwi.4.2.181-188.

[3] Liu X P and Wan C X, Minimizing XML Queries Using a Family of Constraints, In: Proceedings of 4th International Conference on Fuzzy Systems and Knowledge Discovery, Haikou, China. 24-27 August, 2007, IEEE Computer Society, 601-607.

[4] A. Y. Sihem and S. R. Cho, Minimization of Tree Pattern Queries, ACM SIGMOD International Conference on Management of Data, 2001, pp. 497-508.

[5] Prakash Ramanan, Efficient Algorithms for Minimizing Tree Pattern Queries, ACM SIGMOD International Conference on Management of Data, June, 2002.

[6] Wood P. T., Minimising Simple XPath Expression, In: Proceeding of the 4th WebDB International Workshop on the Web and Database, Santa Barbara, California, USA. May 24-25, 2001.

[7] Stelios Paparizos, Yuqing Wu, Laks V. S. Lakshmanan, and H. V. Jagadish, Tree Logical Classes for Efficient Evaluation of XQuery, ACM SIGMOD International Conference on Management of Data, 2004, pp. 71-82.

[8] Jiang H, Lu H, and Wang W, Efficient Processing of XML Twig Queries with OR-Predicates, ACM SIGMOD International Conference on Management of Data, pp. 59-70.

[9] Chan C Y, Fan W, and Zeng Y, Taming XPath Queries by Minimizing Wildcard Steps, In: Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31-September 3, 2004, Morgan Kaufmann, 156-167.

[10] Lu J H, Ling T W, Bao Z F, and Wang Chen, Extended XML Tree Pattern Matching: Theories and Algorithms[J], IEEE Transactions on Knowledge and Data Engineering, 2011, 23(3): 402-416.

Hui Li was born in Qingdao in 1987. She is a M.S. candidate at Beijing University of Technology in P.R.China. Her major field of study is the optimization of Twig query

Husheng Liao was born in Changchun in 1954. He is a professor and doctoral supervisor at Beijing University of Technology in P.R.China. His research interests include software automation methods and data integration technology, etc.

Hang Su was born in Shenyang in 1978. He is a lecturer of computer science at the Beijing University of Technology in P.R.China. His current interests include XML technology, query languages and program transformation.