Mining Positive and Negative Association Rules from XML Query Patterns for Caching

Ling Chen, Sourav S. Bhowmick and Liang-Tien Chia

School of Computer Engineering,
Nanyang Technological University, Singapore, 639798, SINGAPORE

Abstract. Recently, several approaches that mine frequent XML query patterns and cache their results have been proposed to improve query response time. However, frequent XML query patterns mined by these approaches ignore the temporal sequence between user queries. In this paper, we take into account the temporal features of user queries to discover association rules, which indicate that when a user inquires some information from the XML document, she/he will probably inquire some other information subsequently. We cluster XML queries according to their semantics first and then mine association rules between the clusters. Moreover, not only positive but also negative association rules are discovered to design the appropriate cache replacement strategy. The experimental results showed that our approach considerably improved the caching performance by significantly reducing the query response time.

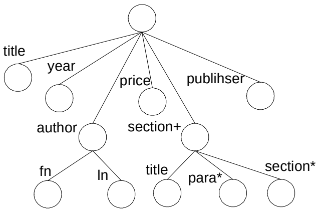
1 Introduction

Extensible Markup Language (XML) has emerged as a standard for data representation and exchange on the World Wide Web. With the rapid growth of XML applications, there is a pressing need to swiftly retrieve information from remote XML sources. Consequently, issues related to efficient processing of XML queries have received considerable attentions.

Recently, caching XML queries has been recognized as an orthogonal approach to improve the performance of XML query engines [3] [11]. Three basic issues are involved in XML query caching: 1) *Containment Relationship*: When a new XML query is issued, decisions should be made whether it is contained by any cached queries so that answers to it can be retrieved from the local cache. 2) *Query Rewriting*: If the new XML query is contained by or overlapping with some cached queries, it should be rewritten with respect to these cached ones. 3) *Replacement Strategy*: A value function should be applied to each query region. When additional space is required in the cache, regions with the lowest values will be the victims. In this paper, we focus on the third problem.

1.1 Motivation

As the cache space is a limited resource, appropriate replacement strategy should be designed to discard data to free space for new data while keep-



(a)

ID	T_1	T_2
1	Q_1 = *book / title*	Q_2 = *book / author*
2	Q_3 = *book [year = " 2004 "] / title*	Q_4 = *book / price*
3	Q_5 = *book / section*	Q_6 = *book / section / title*
4	Q_7 = *book / title*	Q_8 = *book / author / ln*
5	Q_9 = *book [title=" XML"] / author / ln*	Q_{10} = *book / section*

(b)

Fig. 1. DTD and Queries

ing the cache performance. FastXMiner [11] mined frequent XML query patterns from the user queries. Once the cache is full, query regions of infrequent query patterns will be purged first. However, frequent query patterns may not always be reliable in predicting the subsequent queries, as the frequent query pattern-based technique in [11] ignores the temporal feature of user queries. Consider the XML DTD tree in Figure 1 (a) and two sequential queries of five users (expressed as XPath query for ease of exposition) at time points T_1 and T_2 in Figure 1 (b). Applying FastXMiner [11] here will result in the following two cases.

- If we apply FastXMiner at T_1, then we consider the queries in the second column of the table in Figure 1 (b). Suppose the *minimum support* is 0.4. We discover that *book/title* is a frequent query pattern. Unfortunately, caching answers to *book/title* cannot benefit the processing of queries at T_2, as none of the users inquires the information of *book/title*.
- If we apply FastXminer at T_2, then we consider all the queries in the second and third columns of the table in Figure 1 (b). Suppose the *minimum support* is 0.2. We discover that *book/title*, *book/author*, *book/author/ln* and *book/section* are all frequent query patterns. However, if the cache space is not enough to accommodate all these frequent queries, FastXMiner cannot break ties to improve the cache performance.

Hence, in this paper, we consider the sequence between user queries to discover association rules. We use the association rules to predict the subsequent user queries and the confidence of the rules to break ties.

However, few users issue the *exactly* same queries sequentially. For example, consider the queries in Figure 1 (b) again. If only the exactly same queries are considered, then no association rule will be discovered as no two rows are same. Hence, rather than mining association rules between exactly same queries, we mine association rules between semantically related queries. The intuition is that although users may not issue the exactly same queries sequentially, it is possible that they inquire the similar information in sequence. For example, the first and the forth rows in Figure 1 (b) are different in the queries at time T_2. One is *book/author* and the other is *book/author/ln*. Since the two queries are semantically related, we can cluster them into a group representing the queries about the information of book author. Then, an association rule between queries about book title and queries about book author may be discovered from the table in Figure 1 (b). According to this rule, we can predict that users will probably query the information of book author subsequently if they queried the information of book title. Then we can delay the eviction of the information of book author if they are cached before.

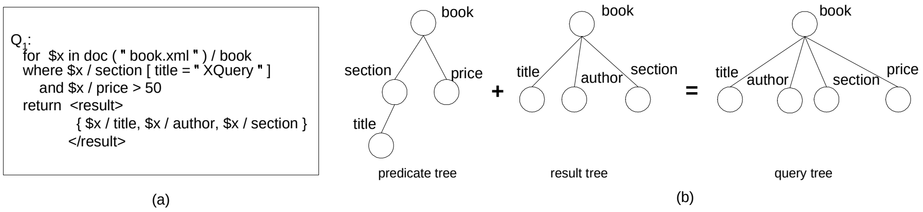
1.2 Overview and Contributions

Firstly, we cluster user queries so that queries about similar information are grouped together. Next, we mine association rules between the clusters. Particularly, we mine association rules between singular query clusters. That is, there is only one cluster on both sides of our association rules. This restriction frees us from maintaining too many historical queries of a user to predict his subsequent query, and significantly reduce the complexity of the mining process. In addition to positive association rules, we also mine negative association rules, which indicate when a user issue some query, she/he probably will not issue some other query subsequently. Finally, we design an appropriate replacement strategy based on the knowledge obtained from the discovered rules.

The main contributions of this paper are summarized as follows.

- We proposed to mine association rules from user queries for XML caching, which is the first that captures the temporal features of user queries to discover knowledge for optimizing caching strategy.
- We designed a novel method to cluster XML queries based on their semantics.
- We implemented our approach and conducted various experiments. Experimental results showed that the replacement strategy incorporated with discovered association rules had better performance than existing approaches.

The rest of the paper is organized as follows. Section 2 briefly discuss some related work of XML query caching in the literature. Sections 3 and 4 present our approach in two stages, clustering user queries based on their semantics and mining association rules. Section 5 shows the experimental results and the comparison with other algorithms. We conclude the paper and outline future directions of research in Section 6.



```
Q1:
 for  $x in doc ( " book.xml " ) / book
 where $x / section [ title = " XQuery " ]
     and $x / price > 50
 return  <result>
             { $x / title, $x / author, $x / section }
           </result>
```

Fig. 2. XQuery Tree

2 Related Work

Due to its flexibility, semantic caching was popular in XML query caching [6] [3]. Hristidis and Petropoulos [6] proposed a compact structure, *modified incomplete tree (MIT)*, to represent the semantic regions of XML queries. ACE-XQ [3] is a holistic XQuery-based semantic caching system. The authors discussed how to judge whether a new query is contained by any cached query and how to rewrite the new query with respect to the cached queries. However, this work did not consider using the knowledge mined from historical user queries to design the replacement function.

Recently, intelligence has been incorporated into Web/XML query caching by constructing predictive models of user requests with the knowledge mined from historical queries [7] [2] [11]. Lan et al. [7] mined association rules from Web user access patterns. Then they prefetched Web documents based on discovered associations and current requested documents. They focused on the placement strategy (fetching and prefetching) while we focused on the replacement strategy. Bonchi et al. [2] mined association rules from Web log data to extend the traditional LRU replacement strategy. However, their work cannot be applied in XML query caching directly because answers to XML query do not have explicit identifiers such as URL. Hence, our work is different from this one in that we mine association rules between query groups in which queries are semantically close. Furthermore, we also use negative association rules to demote the replacement values of corresponding query regions.

3 Query Clustering

Due to the intuition that few users issue the exactly same queries sequentially while many users may inquire similar information consecutively, we cluster the queries based on their semantics before mining association rules. In this section, we discuss our clustering method.

3.1 Clustering Criterion

An XML query can be represented as a node labeled tree. For example, consider the query Q_1 expressed in XQuery syntax in Figure 2 (a). The

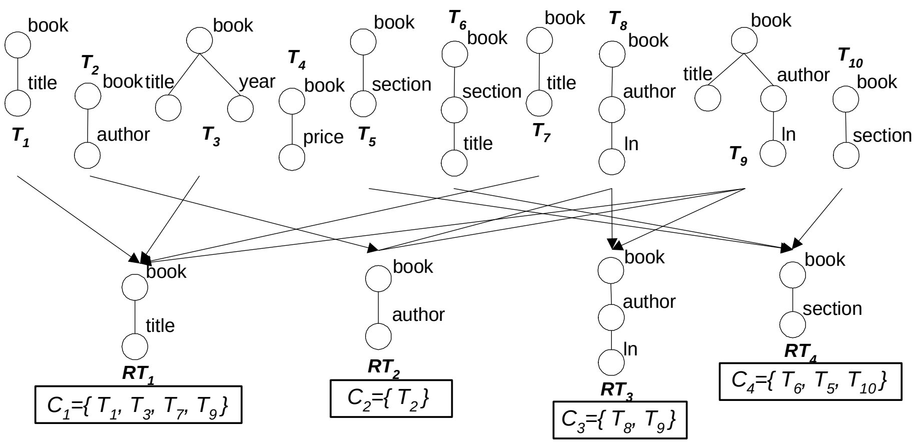


Fig. 3. Initial Clusters

semantics of query Q_1 are composed of two essential parts: the predicate part (*for-where* clauses) and the result part (*return* clause). Both parts can be represented as a tree and the query tree can be constructed by combining the two trees. For example, the query tree of Q_1 in Figure 2 (a) is shown in Figure 2 (b).

After representing each XML query as a query tree, the semantics of the query is captured by its query tree structure. For example, the query tree of Q_1 in Figure 2 (b) indicates that Q_1 inquires the information of the *title, author, section* and *price* of the book. Hence, for the purpose of clustering queries based on their semantics, we can cluster them based on their tree structures.

Existing approaches of clustering tree structures usually employ the agglomerative clustering technique [8] [4]. They are different in defining the *similarity* between two trees or clusters. Basically, the definitions of the similarity can be divided into the following two categories: node-based [4] and edge-based [8]. In order to achieve better accuracy in clustering trees, in this paper, we base our similarity measure on considering the common rooted subtrees between XML query trees. Intuitively, query trees sharing larger common rooted subtree should be semantically closer.

3.2 Clustering Method

Now, we discuss the clustering method. Basically, we cluster query trees by using a "cluster-centered" method [9] [5] rather than an agglomerative method. We employ such a clustering strategy here as it is revealed in [5] that the "cluster-centered" method can distinguish the documents of different semantics better and achieve higher clustering accuracy.

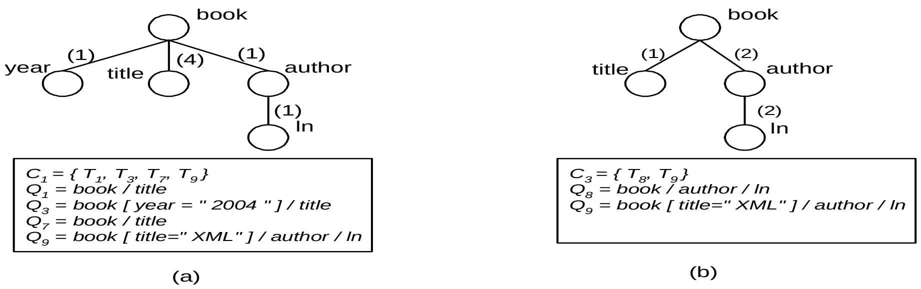

book
year
(1)
title
(4)
(1)
author
(1)
ln
$C_1 = \{ T_1, T_3, T_7, T_9 \}$
Q_1 = book / title
Q_3 = book [year = " 2004 "] / title
Q_7 = book / title
Q_9 = book [title=" XML"] / author / ln
(a)
book
title
(1)
(2)
author
(2)
ln
$C_3 = \{ T_8, T_9 \}$
Q_8 = book / author / ln
Q_9 = book [title=" XML"] / author / ln
(b)

Fig. 4. Intra-cluster Dissimilarity

In order to employ such a clustering method, we should discover the frequent rooted subtrees before clustering user queries. We borrow the algorithm FastXMiner [11] to discover frequent rooted subtrees from XML queries.

Example 3.1 Consider the ten queries in Figure 1 (b), whose query tree structures are redrawn in the upper part of Figure 3. Suppose the *minimum support* δ is 0.2. Four rooted subtrees, *book/title*, *book/author*, *book/author/ln* and *book/section*, are frequent as shown in the lower part of Figure 3.

After discovering frequent rooted subtrees from the collection of query trees, our method constructs clusters in the following three steps: initializing clusters, disjointing clusters and pruning clusters.

Initializing Clusters In this step, we construct the initial clusters. We use the frequent rooted subtrees as the labels of the initial clusters. A query tree will be assigned to an initial cluster if the label of the cluster is the maximal frequent rooted subtree included by the query tree[1]. For example, consider the query T_8 in Figure 3. Two frequent rooted subtree are included by it, RT_2 and RT_3. We assign T_8 to the initial cluster of RT_3 since RT_2 is not the maximal frequent rooted subtree included by T_8. If a query tree does not contain any frequent rooted subtree, such as T_4, the semantic of the query is not significant in the collection of queries and the query tree will be treated as an outlier. Initial clusters may not be disjoint because a query tree may contain more than one maximal frequent rooted subtrees, such as T_9. Thus, T_9 is assigned to the two corresponding initial clusters.

Disjointing Clusters We make the initial clusters disjoint in this step. For each query tree, we identify the best initial cluster and keep the

[1] A frequent rooted subtree is not maximal w.r.t. a query tree if it is included by any other frequent rooted subtree included by the query tree.

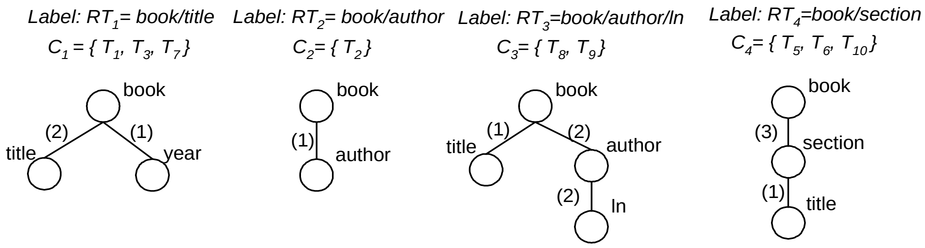

Label: RT_1= book/title
C_1 = { T_1, T_3, T_7}
book
(2)
(1)
title
year
Label: RT_2= book/author
C_2= { T_2}
book
(1)
author
Label: RT_3=book/author/ln
C_3= { T_8, T_9}
book
(1)
(2)
title
author
(2)
ln
Label: RT_4=book/section
C_4= { T_5, T_6, T_{10}}
book
(3)
section
(1)
title

Fig. 5. Clusters after Disjointing

query only in the best cluster. We define the *goodness* of a cluster for a query tree based on the intra-cluster dissimilarity. We measure the intra-cluster dissimilarity based on the number of infrequent edges in the cluster. That is, we merge all query trees in a cluster into a tree structure. For the merged tree, each edge e is associated with a support, denoted as *supp(e)*, which is the fraction of query trees containing it. Given a *minimum cluster support* ξ, an edge in the merged tree is *infrequent* if its support is less than ξ. Then, we define the intra-cluster dissimilarity as follows.

$$Intra(C_i) = \frac{|\{e \in M_i | supp(e) < \xi\}|}{|\{e \in M_i\}|}$$

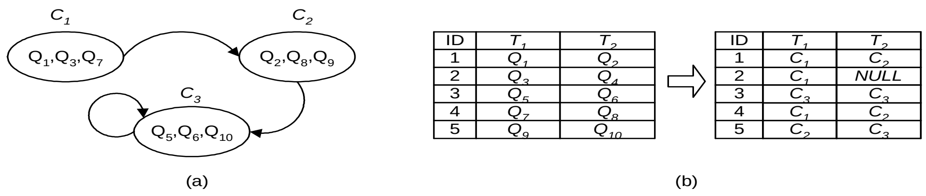
where M_i is the merged tree of all query trees in cluster C_i. The value of *Intra(*C_i*)* ranges from 0 to 1. The higher the *Intra(*C_i*)*, the more dissimilar the query trees in cluster C_i. We assign a query tree to a cluster such that the *intra-cluster dissimilarities* are exacerbated least. That is, a query tree T_i is kept in cluster C_j if

$$C_j \;=\; argmin_{C_j \in C, T_i \in C_j} Intra(C_j)$$

Example 3.2 For example, T_9 is assigned to both initial clusters C_1 and C_3. The merged trees for the two clusters are shown in Figure 4. Let the *minimum cluster support* ξ be 0.6. Grouping T_9 in C_1 generates the *Intra(*C_1*)* $= 0.75$, whereas grouping T_9 in C_3 results in the *Intra(*C_3*)* $=$ 0.66. Hence, we remove T_9 from cluster C_1. After this step, the initial clusters in Figure 3 are adjusted as shown in Figure 5, where each cluster is represented as a merged tree of all query trees in it.

Pruning Clusters If the *minimum support* δ is small, many frequent rooted subtrees will be mined from the user queries. Then there may be many clusters while only some of them are semantically close. Hence, in this step, we perform cluster pruning to merge close clusters.

We measure the inter-cluster similarity based on the number of frequent edges the clusters share. Given a *minimum cluster support* ξ, the



ID	T_1	T_2
1	Q_1	Q_2
2	Q_3	Q_4
3	Q_5	Q_6
4	Q_7	Q_8
5	Q_9	Q_{10}

ID	T_1	T_2
1	C_1	C_2
2	C_1	*NULL*
3	C_3	C_3
4	C_1	C_2
5	C_2	C_3

Fig. 6. Association Mining

set of frequent edges of cluster C_i, denoted as F_{C_i}, are the edges in the merged tree of C_i with their support no less than ξ. That is, $F_{C_i} = \{e|e \in M_i \wedge supp(e) \geq \xi\}$, where M_i is the merged tree structure of cluster C_i. Then, we define the *inter-cluster similarity* as follows.

$$Inter(C_i \rightarrow C_j) = \frac{|\{e|e \in F_{C_i}, e \in F_{C_j}\}| - |\{e|e \in F_{C_i}, e \notin F_{C_j}\}|}{|\{e|e \in F_{C_i}\}|}$$

That is, the more overlap in their frequent edges, the closer the two clusters. The value of $Inter(C_i \rightarrow C_j)$ ranges from -1 to 1. If $Inter(C_i \rightarrow C_j)$ is greater than 0, cluster C_i is semantically close to cluster C_j. We merge two clusters C_i and C_j if not only $Inter(C_i \rightarrow C_j)$ but also $Inter(C_j \rightarrow C_i)$ are greater than 0. Furthermore, we select the cluster label of C_j as the label of the merged cluster if $Inter(C_i \rightarrow C_j) > Inter(C_j \rightarrow C_i)$.

Example 3.3 Consider the clusters C_2 and C_3 in Figure 5 again. Let ξ=0.6. Then, $F_{C_2} = \{(book,\ author)\}$ and $F_{C_3} = \{(book,\ author), (author,\ ln)\}$. Thus, $Inter(C_2 \rightarrow C_3)$ = (1-0)/1 = 1 because the frequent edge in C_2 is frequent as well in C_3. Whereas, $Inter(C_3 \rightarrow C_2)$ = (2-1)/2 =0.5 because the frequent edge (*author, ln*) in C_3 is infrequent in C_2. Hence, we merge C_2 and C_3 and use *book/author/ln* as the cluster label of the new cluster.

The final clustering result of the ten queries in Figure 1 (b) is shown in Figure 6 (a). The semantics of the queries in a cluster can be approximately represented by the cluster label.

4 Association Rule Mining

In this section, we discuss the second stage of our approach: mining positive and negative association rules between the clusters. The input of this stage is the set of *2*-cluster sequences, which is generated by replacing the queries with the corresponding clusters created in the first stage. For example, using the final clustering results as in Figure 6 (a), the initial XML queries in Figure 1 (b) is transformed to a set of five *2*-cluster sequences as shown in Figure 6 (b). Note that, outlier queries, such as query Q_4, are replaced with NULL.

In this paper, we employed the metric *interest* on top of the *support-confidence* framework as in [10] to define positive and negative associ-

Algorithm 1 Positive and Negative Association Rule Generation

```
Input: $D, min\_supp, min\_conf, min\_interest$
Output: $PR$: A set of positive association rules, $NR$: A set of negative association rules
Description:
scan $D$ to find frequent 1-sequence ($F_1$) /*$supp(<C_i,>)$ or $supp(<,C_i>) \geq min\_supp$*/
$P_2 = F_1 \bowtie F_1$ /*candidate frequent 2-cluster sequence*/
for each $<C_i, C_j> \in P_2$ do
  if $supp(<C_i, C_j>) \geq min\_supp$ then
    if ($confidence(C_i \Rightarrow C_j) \geq min\_conf$)&& ($Interest(C_i \Rightarrow C_j) \geq min\_interest$) then
      $PR = PR \cup \{C_i \Rightarrow C_j\}$
    end if
  else
    if $supp(<C_i, \neg C_j>) \geq min\_supp$ then
      if ($confidence(C_i \Rightarrow \neg C_j) \geq min\_conf$)&& ($Interest(C_i \Rightarrow \neg C_j) \geq min\_interest$)
      then
        $NR = NR \cup \{C_i \Rightarrow \neg C_j\}$
      end if
    end if
  end if
end for
```

ation rules. We represent *1*-cluster sequences as $<C_i, >$ or $< ,C_i>$ to distinguish the different positions of cluster C_i. A sequence of clusters $<C_i, C_j>$ supports two *1*-cluster sequences $<C_i, >$ and $< ,C_j>$, and one *2*-cluster sequence $<C_i, C_j>$. Let D be a database of sequences of 2-cluster over $C= \{C_1, ..., C_k\}$. Let $supp(<C_i, C_j>)$ be the fraction of sequences in D that support it, $conf(<C_i, C_j>) = \frac{supp(<C_i,C_j>)}{supp(<C_i,>)}$, *interest*$(<C_i, C_j>)= \frac{supp(<C_i,C_j>)}{supp(<C_i,>)supp(<,C_j>)}$. Given the user defined *minimum support* α, *minimum confidence* β and *minimum interest* γ, $C_i \Rightarrow C_j$ is a positive association rule if 1) $supp(<C_i, C_j>) \geq \alpha$; 2) $conf(<C_i, C_j>) \geq \beta$; 3) $interest(<C_i ,C_j>)) > \gamma$. A negative association rule$C_i \Rightarrow \neg C_j$ can be defined similarly. Instead of discovering frequent *2*-cluster sequences first and then deriving possible rules as commonly done by traditional association rule mining algorithm, we discover positive and negative association rules directly. The algorithm is presented in Algorithm 1.

Finally, we discuss how to design the replacement strategy with discovered association rules. Without loss of generality, we assume that "*the most recent value for clusters*", V_{top}, is incremented by one, each time a new query Q_i is issued. Suppose Q_i is semantically contained by or close to an existing cluster C_i, a positive association rule $C_i \Rightarrow C_j$ with confidence σ was discovered, and the current replacement value of C_j isV_j. Then, we calculate a new replacement value for C_j as $V_j' = V_j + (V_{top} - V_j) \times \sigma$. Since $V_j \leq V_j' \leq V_{top}$, we delayed the eviction of queries in cluster C_j based on the rule. It is similar for negative association rules. For example,

Table 1. Parameter List & Clustering Accuracy

N	Number of query trees	1K-10K
L	Number of potential frequent rooted subtrees	8
P	Maximum overlap between frequent rooted subtrees	0.5
O	The ratio of outliers	0.05
D	Average depth of query trees	4
F	Average fanout of query trees	4

(a)

N	DS	IS
1K	0.026	0.082
2K	0.022	0.080
4K	0.047	0.096
6K	0.038	0.112
8K	0.051	0.128

(b)

with a negative rule $C_i \Rightarrow \neg C_j$, we update $V_j' = V_j+(V_j\text{-}V_{top})\times\sigma$. As $V_j' < V_j$, we actually hasten the the purge of queries in cluster C_j.

5 Performance Study

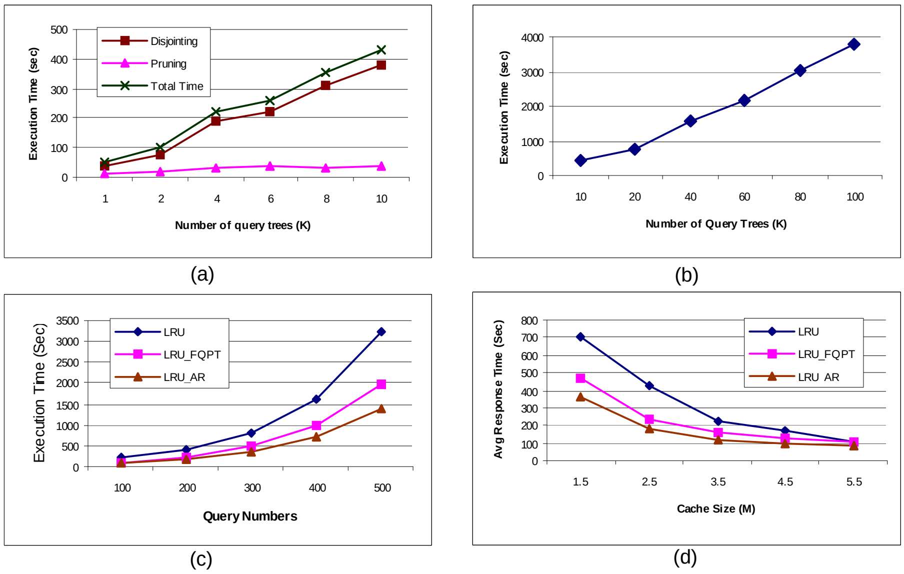
In this section, we evaluate the performance of our approach with some preliminary experimental results. We implemented our approach in Java. Experiments are carried out on a Pentium IV 2.8GHz PC with 512 MB memory. The operating system is Windows 2000 professional.

5.1 Performance of Query Clustering

Firstly, we investigate the performance of our query clustering method. Given a DTD file, We generate synthetic query trees in the following steps: 1) A set of potential frequent rooted subtrees is generated by controlling the overlap between them. 2)We also create some infrequent rooted subtrees as outliers. 3) Finally, we generate the query trees based on the rooted subtrees produced in the first two steps. The parameters we used in the data set generation process are summarized in Table 1 (a), where the third column shows the default values.

We conducted experiments to evaluate the accuracy, efficiency and scalability of our clustering method respectively by varying different parameters.

- *Accuracy Study.* We evaluate the accuracy of our clustering method by varying the number of query trees from 1,000 to 8,000. The *minimum global support* is set as 5% and the *minimum cluster support* is set as 25%. Table 1 (b) shows the average intra-cluster dissimilarity (DS) and the average inter-cluster similarity (IS) of the resulting clusters. We observed that our clustering method can achieve small intra-cluster dissimilarity and small inter-cluster similarity.
- *Efficiency Study.* We evaluated the efficiency of the clustering method by showing the time cost of different phases in Figure 7 (a) except the cost of initialization step which is very trivial. The main cost of our clustering method is the disjointing step as it recursively optimizes the intra-cluster dissimilarity.



(a)

(b)

(c)

(d)

Fig. 7. Performance of Clustering

- *Scalability Study.* We evaluate the scalability of the clustering method by duplicating the query trees until we get 100K query trees. Figure 7 (b) shows that our clustering algorithm scales well with respect to the number of query trees.

5.2 Performance of Replacement Strategy

We then show the effectiveness of our replacement strategy with discovered association rules. We used a simple XQuery processor [1] that process queries directly from the source XML file. In our experiment, we generated a fragment of DBLP data as the source XML document. The file size is 10.5M and there are totally 248,215 nodes. We first generate a training data set of *2*-cluster sequences to discover positive and negative association rules. Then, we generate a testing data set of *2*-cluster sequences to evaluate the performance of caching.

Two sets of experiments were carried out to investigate the effect of varying the number of queries and varying the size of cache respectively. We compared our association rule based LRU replacement strategy (LRU_AR) with another two strategies, LRU and LRU with frequent query patterns mined by [11] (LRU_FQPT). We use the *Average Response Time*, which is the ratio of total execution time for answering a set of queries to the total number of queries in this set, as the metric.

- Variation of Query Numbers. Because of the limited power of the query processor, we vary the number of queries from 100 to 500 and the cache size is fixed at 2.5MB. As we can see from Figure 7 (c), when the number of queries is large, the average response time of LRU_AR surpasses the other two strategies.

- Variation of Cache Size. We vary the size of cache from 1.5M to 5.5M, and the number of queries is fixed at 300. As shown in Figure 7 (d), the more limited the cache size, the greater gap in average response time between LRU_AR and the other two competitors.

6 Conclusions

In this paper, we presented an approach that mines association rules from XML queries for caching. Since our association rules address the temporal sequence between user queries, it is more reliable in predicting future queries than the approaches that address the frequency or recency only. Due to the intuition that few users issue the exactly same queries sequentially, we cluster queries based on their semantics first and then discover the positive and negative associations between them. The knowledge obtained from the discovered rules are incorporated in designing appropriate replacement strategies. As verified by the experimental results, our approach improved the cache performance significantly.

References

1. http://www.cs.wisc.edu/ mcilwain/classwork/cs764/.
2. F. Bonchi, F. Giannotti, C. Gozzi, and G. Manco et al. Web log data warehousing and mining for intelligent web caching. In *Data and Knowledge Engineering, 39(2):165-189*, 2001.
3. L. Chen, E. A. Rundensteiner, and S. Wang. Xcache-a semantic caching system for xml queries. In *Demo in ACM SIGMOD*, 2002.
4. T. Dalamagas, T. Cheng, K. Winkel, and T. K. Sellis. Clustering xml documents by structure. In *Proc. of SETN*, 2004.
5. B. C. M. Fung, K. Wang, and M. Ester. Hierarchical document clustering using frequent itemsets. In *Proc. of SDM*, 2003.
6. V. Hristidis and M. Petropoulos. Semantic caching of xml databases. In *Proc. of the 5th WebDB*, 2002.
7. B. Lan, S. Bressan, B. C. Ooi, and K. L. Tan. Rule-assisted prefetching in web-server caching. In *Proc. of ACM CIKM*, 2000.
8. W. Lian, D. W. Cheung, N. Mamoulis, and S. Yiu. An efficient and scalable algorithm for clustering xml documents by structure. In *IEEE TKDE, vol. 16, No. 1*, 2004.
9. K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *Proc. of ACM CIKM*, 1999.
10. X. Wu, C. Zhang, and S. Zhang. Mining both positive and negative association rules. In *Proc. of ICML*, 2002.
11. L. H. Yang, M. L. Lee, and W. Hsu. Efficient mining of xml query patterns for caching. In *Proc. of 29th VLDB*, 2003.