Efficient XML Retrieval Service with Complete Path Representation

Hsu-Kuang CHANG†,††(a), Student Member, King-Chu HUNG†, and I-Chang JOU††, Nonmembers

SUMMARY Compiling documents in extensible markup language (XML) increasingly requires access to data services which provide both rapid response and the precise use of search engines. Efficient data service should be based on a skillful representation that can support low complexity and high precision search capabilities. In this paper, a novel complete path representation (CPR) associated with a modified inverted index is presented to provide efficient XML data services, where queries can be versatile in terms of predicates. CPR can completely preserve hierarchical information, and the new index is used to save semantic information. The CPR approach can provide template-based indexing for fast data searches. An experiment is also conducted for the evaluation of the CPR approach.

key words: XML, DTD, complete path representation (CPR), structural summary tree (SST), versatile query

1. Introduction

By recommendation of the World Wide Web Consortium (W3C) [1], XML has served as a standard information description language widely used in computer communication. Both the number and categories of XML documents are rapidly increasing. Today, people are increasingly dependent on fast data services on the internet. This facility demands skillful XML data representation for fast internet searches [2]. Since each XML document can be simplified into a structural summary tree (SST), SST representation can save a significant amount of the space complexity of a great amount of XML data. SST is a unique semantically structured tree that involves hierarchical structure and relationships among nodes; both structure and semantics are significant features for XML data representation.

XML data representations can be categorized into string [3], [4] and path [5]–[15] groups. String representation can be derived with a preorder traversal algorithm; this requires dynamic programming for edit distance measurement. This approach, with its lack of structure information, may lead to indeterminate search results (ambiguity). Path approaches use sub-paths as the features, and represent each XML datum with a binary vector. An element in the binary vector denotes whether the datum involves a corresponding feature, where such features can be defined as single node paths [6], two-node paths (i.e., node-pair (NP)) [7], [8], or whole paths (WP) [9], [10]. To improve search efficiency, several modifications of the path approach have been proposed. Yang et al. [11] used the content instead of the leaf node for node representation. Liu et al. [12] presented a hybrid definition that combines NP and WP for XML data description. Based on the determined finite automata, Mustafa et al. [13] and Lee et al. [14] presented a path-embedded string representation. For the improvement of efficiency in common XPath and principal component analysis, Li [15] presented a modified WP with limited-length paths.

In order to better facilitate data access on the internet, it is desirable to provide versatile data requesting with queries for users, e.g., single node path, NP path, WP path, node-relationship path and compound paths, in terms of predicates. The traditional schemes mentioned above are only concerned with serving the simple cases of NP and WP queries. Recently, based on XML QBE [16], XQuery [17], PathStack and TwigStack [18], and query pattern tree (QPT) [19], several stack-based parsers [20]–[23] have been proposed. These methods translated served objects into an algorithm and directly parsed the original query path instead of matching SST elements. Stack-based parsers can effectively save space complexity, but can only serve limited node-relationship paths, e.g., ancestor-descendant (AD), NP and WP paths. Moreover, both stack-based and NP schemes are inherently ambiguous in serving the SST query paths.

In this paper, based on the concept of preserving complete tree information, a novel XML data representation, referred to as complete path representation (CPR), is proposed. CPR uses complete path elements (CPEs) as features, where CPEs are defined as the overall sub-paths starting from each level of the SSTs. The SST is the simplified tree of an XML datum with nested nodes and repeated branches removed. A modified inverted index structure for recording the semantically structural information of CPEs is also presented. The new tree data structure can provide template-based indexing for fast data searches, which can effectively reduce the searching complexity. Eleven compound-type queries were designed for the evaluation of CPR performance.

This paper is arranged as follows: In Sect. 2, the CPR is proposed for XML data description. In Sect. 3, the modified inverted index is described. Section 4 shows the performance evaluation results of handling versatile queries by
using variant approaches. Finally, discussion and conclusion are drawn in Sect. 5.

2. XML Data Representation Using Complete Path Elements

Any XML datum defined with the document type definition (DTD) can be modeled as an ordered label tree [3]. In this section, the hierarchical tree information is extracted by a pre-ordered traversal process performed with a document object model (DOM) API [1]. Following this, the CPE extraction based on the SSTs of XML data is described and the CPR is presented.

2.1 Structural Summary Tree Extraction of XML Documents

SST is the XML tree skeleton commonly used for XML data representation [3]. The SST of an XML document can be extracted by four functional processes, as follows:

Step 1. This step performs a tree conversion using Java DOM (JDOM), where the tree element values are neglected. For the DTD-formatted XML datum of Example 1, the tree conversion process result is illustrated in Fig. 1.

Step 2. For efficient matching, we symbolize the name of the tree node with an abbreviated character order, as shown in Fig. 2.

Step 3. With the pre-order traversal process [3], SST extraction is based on two simplification procedures:
   a) For each node, examine whether the current node’s name is equal to an ancestor’s name. If it is, set the current node’s sub-tree to be a child of the ancestor; otherwise, check the next node. This procedure serves to remove nested sub-paths, as shown in Fig. 3.
   b) Exhaustive searching based on a Hash table is applied for discovering and eliminating repeated branches. Figure 4 shows the repeated branch elimination result where the simplified tree is the SST.

Step 4. For the extraction of CPEs, it is necessary to construct the adjacent-linked (AL) lists of the SSTs of all XML data. An AL list is a data structure that records the linking information of each node and facilitates the pre-order traversal process. The AL list of the SST in Fig. 4 is given in Table 1 where \( \delta_i[n] \) denotes the n-th head node of the i-th XML datum.

Example 1.

\[
\begin{align*}
\text{<!ELEMENT books (books*, intro*)+>} \\
\text{<!ELEMENT intro (title, author)>} \\
\text{<!ELEMENT title (#PCDATA)>} \\
\text{<!ELEMENT author (firstname, lastname)>} \\
\text{<!ELEMENT firstname (#PCDATA)>} \\
\text{<!ELEMENT lastname (#PCDATA)>}
\end{align*}
\]

Table 1 The AL list of the SST of Example 1.

| Head Node | AL list |
|-----------|---------|
| \( \delta_0[0] \) | \( \langle B \rangle \rightarrow \langle 1 \rangle \) |
| \( \delta_0[1] \) | \( \langle 1 \rangle \rightarrow \langle T \rangle \) \( \rightarrow \langle A \rangle \) |
| \( \delta_0[2] \) | \( \langle T \rangle \rightarrow \langle F \rangle \) \( \rightarrow \langle L \rangle \) |
| \( \delta_0[3] \) | \( \langle F \rangle \rightarrow \langle L \rangle \) |
| \( \delta_0[4] \) | \( \langle F \rangle \rightarrow \langle L \rangle \) |
| \( \delta_0[5] \) | \( \langle 1 \rangle \rightarrow \langle N \rangle \) |
2.2 Complete Path Element Extraction and Representation

Traditional WP and NP representations, with their lack of linking information, cannot serve such queries as 
\((/B/*/*/*/*)\) and 
\((/*/*/I/*/*/*/*/L)\), where * means “don’t care” about the node name. For an efficient query service, CPR describes XML data with the CPEs of all SSTs. The CPE set of a tree is defined as all the branches, (i.e., sub-paths) starting from each level and ending with the leaves. For convenience, let \(CPL_{i}\) denote a set of CPEs starting from the \(i\)th level, where the root level is defined as one, and is increased toward the leaves. An example of level definition is shown in Fig. 4, where four CPE sets, \(CPL_{1}, CPL_{2}, CPL_{3},\) and \(CPL_{4}\), can be defined. The elements of the four CPE sets are shown in Table 2.

Based on the AL list, a CPE extraction algorithm using the depth-first search is developed to trace all branches, starting from each level. This algorithm can preserve the order of left and right child nodes. The CPE extraction algorithm is described in Tables 3 and 4, respectively. Inferring from the CPE, the ancestor-descendant (AD), sibling (SB) and cousin (CN) extraction algorithms are described in Tables 5 and 6, respectively.

Essentially, this algorithm is an exhaustive search that is guaranteed to find all of the branches of a SST. The CPR for the description of SSTs can be defined as:
\[
CP_3 = \bigcup_{i=1}^{L} CP_{L-i} \quad CP_{L-i} \text{ is a set involving the } i\text{-th level CPEs of all XML data}
\]

where \( \cup \) denotes union operation and \( L \) is the maximum level number. Considering a database comprised of the three XML data shown in Fig. 5, the CPR can be found in Table 7. In Fig. 5, there are two I nodes for both DOC 1 and 3. The two nodes with different children are distinct and cannot be merged. The two sub-paths /B/I/T in DOC 2 and /B/I/T/**

Table 6  The Siblings (SBs) and Cousins (CNs) build algorithms.

![Diagram](image)

\[ FV_{DOC} = [\rho_0, \rho_1, \cdots, \rho_{N-1}], \quad \rho_i \in \{0, 1\}, \quad (1) \]

where \( \rho_i \) denotes the label of the \( i \)-th CPE, and \( N \) denotes the total number of CPEs. The element with \( \rho_i = 1 \) implies that the document involves the \( i \)-th labeled CPE. With the FV description, CPEs can be labeled with a hierarchical structure, and in DOC 1 have the same path length equal to 3, but have distinct distances from the leaf node. The same distinctions also exist between the two sub-path elements /M/I/** and /M/I/**/* in the \( CP_{L-i} \).

### 3. Indexing Complete Path Elements

A CPE with the tree characteristic is a high dimensional feature. Traditional B-tree indexing [24], [27], [28] based on node relationships is suitable for WP, NP and twig queries, but is inefficient for CPR, which regards each CPE as a feature element. In this section, a new index with feature similarity structure (FSS) is presented for CPE management. The FSS provides fast template-based hierarchical indexing.

The CPEs of Table 7 can be represented with a tree structure, as shown in Fig. 6, where \( \Pi \) denotes a CPE subset with path length equal to \( i \). The CPEs in Fig. 6 are inherent with the hierarchical information involving path length (\( \Pi \)) and level (\( CP_{L-i} \)) that are available for inferring semantic relations, e.g., ancestor-descendant (AD), sibling (SB) and cousin (CN) relationships. A B-tree index with a key design can achieve a balanced binary tree structure for efficiently indexing the elements of NP and WP, but cannot provide hierarchical information. To facilitate the inference of semantic information, the inverted index structure with additional fields is applied for CPE indexing. These additional fields are used for recoding nodes’ children and the CPR level defined in the tree representation of Fig. 6. The modified inverted index referred to as the FSS is defined in Table 8.

As shown in Table 9, the FSS can be used to define either an internal node or a leaf node. The difference between the two data structures is the setting of the active field.

The FSS with feature similarity provides a template-based hierarchical query service. This service method can effectively reduce the searching complexity induced by the path element increment of CPR, compared to that of the NP and WP. Utilizing the one-to-one property of \( \rho_i \), XML documents can be uniquely described with a feature vector (FV), defined as

\[
FV_{DOC} = [\rho_0, \rho_1, \cdots, \rho_{N-1}], \quad \rho_i \in \{0, 1\},
\]

where \( \rho_i \) denotes the label of the \( i \)-th CPE, and \( N \) denotes the total number of CPEs. The element with \( \rho_i = 1 \) implies that the document involves the \( i \)-th labeled CPE. With the FV description, CPEs can be labeled with a hierarchical structure,
The index structure of the tree representation of Fig. 5. Italics denotes a new path inserted.

Table 8 The node structure of FSS for CPE indexing.

| Field                  | Description                                                                 |
|------------------------|----------------------------------------------------------------------------|
| field FSSElem          | Pointer points to node FSSElem                                             |
| 1. struct FSSElem {    |                                                                             |
| 2. int Children;       | Number of children of this node                                           |
| 3. int Descendants;    | Number of descendants                                                     |
| 4. int CPR_Level;      | Level of this node in the tree representation                             |
| 5. int Update_Count;   | Record of modification times                                              |
| 6. String Element;     | Element name of this node                                                 |
| 7. union PointerToChild|                                                                             |
| 8. FSSElem *Child_Array_Pointer (Children) ; Child Pointer            |                                                                             |
| 9. XML_Listings *DocPointer; | Document pointer               |
| 10. );  union PointerToChild;} |                                                                            |
| 11. }; struct FSSElem  |                                                                             |

as shown in Table 10. This labeling provides a template-based hierarchical query service. Let CPsT(i, j) be an indexing template involving the CPEs of CP L−1 and P i. An indexing template with (i, j) = (1, 1) can be defined with:

$$\text{CPsT}(1, 1) = \begin{bmatrix} SW & 0 & 0 & 0 & 0 \\ \rho_0 & \rho_1 & \rho_2 & \rho_3 & \rho_{P_i} \end{bmatrix}_{CP_{L−1}}$$

where SW denotes a switch. Setting a field of SW to one indicates that the corresponding CPE is selected. For example, an indexing template defined by:

$$\text{CPsT}(1, 4) = \begin{bmatrix} SW & 1 & 0 & 1 & 1 \\ \rho_0 & \rho_{13} & \rho_{14} & \rho_{15} & \rho_{16} & \rho_{P_i} \end{bmatrix}_{CP_{L−1}}$$

will yield a response as:

$$\rho_{13} = /B/I/T/D$$ in Doc1

$$\rho_{15} = /B/I/A/L$$ in Doc1 and Doc2

$$\rho_{16} = /M/I/A/L$$ in Doc3.

Like the Region [22] and Dewey [26] methods, the CPE index can be easily updated with numerical labeling, as shown in Table 11. Updating the Dewey method is based on the extended Dewey labeling [25], [27], [28] which uses modular function to reserve even numbers for the insertion of new path elements. On the other hand, updating the CPE index only requires increasing the label in a template. Suppose that a new CPE /B/I/A/M will be added between /B/I/A/F and /B/I/A/L, as shown in Fig. 6. This updating will introduce four new CPEs: /CP L−1/P i/4/*/A/M, /CP L−3/P i/3/*/A/M, /CP L−3/P i/2/*/A/M and /CP L−4/P i/2/*/A/M (the italic type in Fig. 6), and lead to some modifications: CPsT(1, 4), CPsT(2, 3), CPsT(3, 2) and CPsT(4, 1), as shown in Table 11, where only the content’s order of CPsT(1, 4) needs to be rearranged (i.e., the new labels in parentheses).

The FSS with path length and level also allows the inference of semantic information. The path elements with AD relationships can be easily obtained from the CPEs with the path length field filled in P i for i ≥ 3, i.e., path length ≥ 3. For the example of Fig. 5, there are two kinds of AD relationship shown in Table 12, where A1 involves the path elements with one-generation AD, and A2 involves the path elements with two-generation AD. Note that these path elements are different from CPE, and are labeled with δ0~δ11. SB and CN are relations among nodes, where these nodes have different descendants, but have the same father and grandfather node, respectively. For SB, the father nodes can be found in levels CP L−1 for 1 ≤ i ≤ L − 1. Furthermore, the search of CN nodes aims to verify whether their father nodes are inherent with a SB relationship. The hierarchical labeling templates of SB and CN relations are shown in Table 13. The tree structure index, including semantic information, is illustrated in Fig. 7, where SB and CN indexing requires fewer levels than the indexing of AD.
Table 10 A hierarchical labeling for the 34 CPEs of Fig. 6.

| CPEs | Path | P1 | P2 | P3 | P4 |
|------|------|----|----|----|----|
| CPE_{L,1} | Path | p # | Path | p # | Path | p # | Path | p # |
| \(*\|T\|{}^*\|T\) | p_0 | 1.1 | \(*\|T\|{}^*\|T\) | p_1 | 1.2 | \(*\|T\|{}^*\|T\) | p_2 | 1.3 |
| \(*\|A\|{}^*\|T\) | p_3 | 1.4 | \(*\|A\|{}^*\|T\) | p_4 | 1.5 | \(*\|A\|{}^*\|T\) | p_5 | 1.6 |

Table 11 A mapping for numerically labeling the 34 CPEs of Fig. 6.

| CPEs | P1 | P2 | P3 | P4 |
|------|----|----|----|----|
| CPE_{L,1} | /CPEs/CPE_{L,1}/P1 | /CPEs/CPE_{L,1}/P2 | /CPEs/CPE_{L,1}/P3 | /CPEs/CPE_{L,1}/P4 |
| p # | Labels | p # | Labels | p # | Labels | p # | Labels |
| p_0 | 1.1 | p_1 | 1.2 | p_5 | 1.3 | p_7 | 1.4 |
| p_1 | 1.2 | p_1 | 1.2 | p_6 | 1.3 | p_8 | 1.4 |
| p_2 | 1.3 | p_6 | 1.3 | p_9 | 1.4 | p_10 | 1.5 |
| p_3 | 1.4 | p_7 | 1.4 | p_11 | 1.5 | p_12 | 1.6 |

Table 12 The template-based hierarchical labeling for the AD path elements of Fig. 5.

| ADs | A1 | A2 |
|-----|----|----|
| AD_{L,1} | Path | δθ | Path | δθ |
| \(*\|T\|{}^*\|T\) | δ_1 | \(*\|T\|{}^*\|T\) | δ_1 |
| \(*\|A\|{}^*\|T\) | δ_1 | \(*\|A\|{}^*\|T\) | δ_1 |
| \(*\|T\|{}^*\|A\|{}^*\|T\) | δ_2 | \(*\|T\|{}^*\|A\|{}^*\|T\) | δ_2 |

Table 13 The path elements with SB and CN relations for Fig. 6.

| SBs | CNs |
|-----|-----|
| SB_{L,2} | Path | p # | CN_{L,3} | Path | p # |
| \(*\|T\|{}^*\|T\) | p_0 | \(*\|T\|{}^*\|T\) | p_2 |
| \(*\|A\|{}^*\|T\) | p_1 | \(*\|A\|{}^*\|T\) | p_2 |
| SB_{L,3} | Path | p # | CN_{L,4} | Path | p # |
| \(*\|T\|{}^*\|T\) | p_3 | \(*\|T\|{}^*\|T\) | p_3 |
| SB_{L,4} | Path | p # | CN_{L,5} | Path | p # |
| \(*\|T\|{}^*\|T\) | p_4 | \(*\|T\|{}^*\|T\) | p_4 |
Fig. 7 The index structure of the AD, SB, and CN relationships of Fig. 6.

Table 14 The abbreviated node name of movie posts.

| MOVIES Element Name | :: abbreviation |
|---------------------|-----------------|
| W4F DOC :: W4FD     | Movie :: Mv     |
| Year :: y           | Director :: Dr  |
| Genres :: Gns       | Cast :: Cst     |
| Actor :: Act        | FirstName :: FN |
|                     | LastName :: LN  |

Table 15 The abbreviated node name of the PP group.

| ProceedingsPage Element Name | :: abbreviation |
|------------------------------|-----------------|
| ProceedingsPage :: PP        | Location :: Loc |
| conference :: c              | year :: y       |
| month :: m                   | number :: n     |
| volume :: v                  | sectionList :: SL|
| sectionList Tuple :: SLT     | sectionName :: SN|
| articleTuple :: arcsT        | Toindex :: Ti   |
| href :: hf                   | index :: idx    |
| xml:link :: x1               | inline :: il    |
| endPage :: ep                | initPage :: ip  |
| authors :: authors           | author :: auth  |

Table 16 The abbreviated node name of the ITP group.

| IndexTermsPage Element Name | :: abbreviation |
|-----------------------------|-----------------|
| conName :: cn               | number :: n     |
| initPage :: ip              | Abstractor :: ab|
| id :: id                    | Size :: s       |
| inline :: il                | generalTerms :: GTs |
| term :: tm                  | CategoryAndSubjectDescriptors :: CASDT|
| Content :: cnt              |                 |

Fig. 8 The SST of the dataset MOVIES.

Fig. 9 The SST of the PP group in the ACM.

Fig. 10 The SST of the ITP group in the ACM.
From the above description, the establishment procedure of the CPR index can be summarized as follows:

a. Construct the SST and its AL list based on the four steps described in Sect. 2.
b. Extract CPEs with BuildCprAlgorithm and DwsCPElements in Tables 3 and 4, respectively.

c. Extract AD, SB, and CN elements, based on the CPEs and semantic information (i.e., path length and level) in Tables 5 and 6, respectively.
d. Declare the data structure of CPR nodes (i.e., the FSSele in Table 8).
e. Construct the CPR index tree (i.e., Figs. 6 and 7).
f. Build the Query template based on the information of level and path length.
g. Finally, Build the updating function of the CPR index tree with numerical labels.

### 4. Experimental Results

For performance evaluation of the CPR approach, two datasets named ACM and MOVIES were built with the XML documents downloaded from the INEX Record homepage [29]. The MOVIES dataset comprises 489 movie posts of action/adventure/war, comedy/drama, science fiction, horror/mystery/thriller, musical, and others. Note that all movie posts can be described with only one SST due to similar XML structure. The ACM dataset contains 937 XML documents, which can be categorized into two DTD groups, ProceedingsPage.dtd (PP) and IndexTermsPage.dtd (ITP), where the PP group has 17 documents and the ITP group has 920 documents. Using the abbreviation defined in Tables 14–16, the SSTs of the three groups are given in Figs. 8–10. The CPR of the PP group has 8 levels and 6-generation AD relationships. The two groups, MOVIES and ITP, have many more documents, but shorter SSTs than the PP group. The CPRs of the two groups have 5 and 4 levels, respectively.

For simulating versatile queries, eleven XML data request queries were designed in Table 17. With the query parser [22]–[25], these query statements can be translated into compound tree-pattern queries (TPQs). The 11 TPQs can be partitioned into simple path (TPQ1~TPQ5), relationship (TPQ6~TPQ9) and combination (TPQ10~TPQ11) groups. Service efficiency analysis is performed by a measurement of searching complexity (SC). The SC index is defined with the number of node comparisons required for the path matching between query and database. Let Ndp denote the required node number of database pad in the path matching. Table 18 shows the SC definitions of WP, NP and CPR schemes, where the Ndp of CPR can be variant due to the use of different searching templates. Since B-tree index does not provide path length information, we suppose that the matching of each WP path needs to check all WP path nodes in the database pad. Thus the Ndp of the WP scheme (NdpWP) is defined with all the path nodes in the database pad and the Ndp of the hybrid scheme (NdpHB) is defined with the summation of NdpWP and NdpNP. For the three heterogeneous XML datasets, the Ndp values of variant representation schemes are shown in Table 19, where the entry 20:97 in the NdpWP column denotes that there are 20 WP path elements that totally involve 97 nodes in the PP database pad. The NdpCPS entry 12:12 of the PP group denotes that the template CPS(T,2) involves 12 nodes (i.e., 12 path elements with each path having one node). The NdpADs entry 7:14 in the location (ADL−1,A5) denotes that the template ADsT(1,5) involves 7 path elements, and each path has two nodes with 5 generation AD relationships.

The path numbers (n) and path lengths (α and β) of the

---

**Table 18** Formulas of SC and Ndp for different schemes.

| Schemes | SC Formula | Parameters |
|---------|------------|------------|
| WP | $SC_{WP} = \sum_{i=1}^{N} \alpha_i \times Ndp_{WP}$ | $n$: numbers of WP searching paths, $\alpha_i$: nodes of the i-th WP searching path |
| NP | $SC_{NP} = \sum_{i=1}^{N} \beta_i \times Ndp_{NP}$ | $n$: numbers of NP searching paths, $\beta_i$: nodes of the i-th NP searching path |
| HB | $SC_{HB} = SC_{WP} + SC_{NP}$ | $SC_{HB}$: combination of SCWP and SCNP |
| CPR | $SC_{CPR} = \sum_{i=1}^{n} \rho_i \times Ndp_{CPR,TBI}$ | $n$: numbers of CPR searching paths, $\rho_i$: CPR searching paths, $Ndp_{CPR,TBI}$: CPR template based index (TBI) |

| Schemes | Ndp Formula | Parameters |
|---------|------------|------------|
| WP | $Ndp_{WP} = \sum_{i=1}^{N} l_i$ | $N$: numbers of paths in NdpWP, $l_i$: path length of the i-th WP |
| NP | $Ndp_{NP} = \sum_{i=1}^{N-1} Np_i$ | $Np_i$: path length of the i-th NP |
| HB | $Ndp_{HB} = \sum_{i=1}^{N} l_i + \sum_{i=1}^{N} Np_i$ | Combination of NdpWP and NdpNP |
| CPR | $CPEs$ | $Ndp_{CPEs} = \sum_{i=1}^{L} \sum_{j=1}^{\delta_{i,j}^{L}} \beta_{i,j}$ | $L$: max level of SST, $\delta_{i,j}^{L}$: i-level, j-length complete paths |
| | $ADs$ | $Ndp_{ADs} = \sum_{i=1}^{L-2} \sum_{j=1}^{\delta_{i,j}^{L-2}} \delta_{i,j}$ | $L$: max level of SST, $\delta_{i,j}^{L}$: i-level, j-generation AD paths |
| | $SBs$ | $Ndp_{SBs} = \sum_{i=1}^{L} \phi_{i}$ | $L$: max level of SST, $\phi_{i}$: one-node sibling paths |
| | $CNs$ | $Ndp_{CNs} = \sum_{i=1}^{L} \lambda_{i}$ | $L$: max level of SST, $\lambda_{i}$: one-node cousin paths |
Table 19  The Ndp values for PP, ITP, and MOVIES datasets.

| XML  | $Ndp_{PP}$ (path: Ndp) | $Ndp_{ITP}$ | $Ndp_{MOVIES}$ |
|------|------------------------|-------------|----------------|
| PP   | 20: 97                 | 31: 62      | 51: 159        |
| ITP  | 15: 40                 | 20: 40      | 35: 80         |
| MOVIES | 6: 24              | 13: 26      | 19: 50         |

Table 20  A query service performance comparison of four XML data representation schemes.

| Tree Pattern Queries | SC value |
|----------------------|----------|
|                      | SC$_{WP}$ | SC$_{NS}$ | SC$_{IHP}$ | SC$_{CPR}$/Template |
|                      | n        |       | n          | n          |                      |
| WP [8-10]            | 2        | 4     | -          | -          | 192                 |
| TPQ 1                | 2        | 4     | 2          | 4          | 192                 | 96:CPsT(1,4)       |
| TPQ 2                | -        | -     | 3          | 2          | 156                 | 36:CPsT(1,2)       |
| TPQ 3                | -        | -     | 3          | 2          | 244                 | 108:CPsT(1,4),CPsT(1,2) |
| TPQ 4                | -        | -     | 2          | 2          | 248                 | 32:CPsT(2,2)       |
| TPQ 5                | -        | -     | 2          | 2          | 160                 | 56:CPsT(2,2)       |
| TPQ 6                | -        | -     | -          | -          | -                   | 64:CPsT(2,4)       |
| TPQ 7                | -        | -     | 2          | 2          | 2                   | 70:ADsT(2,1),ADsT(2,2) |
| TPQ 8                | -        | -     | -          | -          | -                   | 10:SBsT(3)         |
| TPQ 9                | -        | -     | 2          | 2          | -                   | 6:CNsT(4)          |
| TPQ A                | -        | -     | 4          | 2          | 2                   | 88:ADsT(5,1),SBsT(7),CNsT(7) |
| TPQ B                | -        | -     | 4          | 2          | 2                   | 92:ADsT(5,1),CNsT(7) |

* : TPQ1 has two paths (n=2), the length of each path is 4, so SC$_{WP}$ = 4*24+4*24 = 192. This path is indexed in the template CPsT(1,4), thus SC$_{CPR}$ = 4*12+4*12 = 96.

** : TPQ 3, a hybrid case, has two whole paths and one node pair (NP) path, so SC$_{IHP}$ = 4*24+4*24+2*26 = 244. For CPR, the two whole paths are indexed in CPsT(1,4) and the NP path is indexed in CPsT(1,2), thus SC$_{CPR}$ = 4*12+4*12+2*6 = 108.

*** : TPQ 4 has two NP paths, so SC$_{WP}$ = 2*62+2*62 = 248 where /T/h is an ambiguous path for NP representation scheme. For CPR, the query can be served with the template CPsT(2,2), thus SC$_{CPR}$ = 2*8+2*8 = 32.
Thus the path */t/id in Figs. 9 and 10 are also undistinguishable. The two NP paths are undistinguishable in NP scheme due to lacking level information. Similarly, the two NP path elements */t/id and */*/t/id in Figs. 9 and 10 are also undistinguishable. This approach can effectively eliminate the mapping requirement of saving templates, but it requires a mapping function to transform the three-information into an attribute. This will be one of our future research areas in this subject.

References

[1] World Wide Web Consortium. The document object model. http://www.w3.org/DOM/
[2] T. Dalamagas, T. Cheng, K.J. Winkel, and T. Sellis, “A methodology for clustering XML documents by structure,” Inf. Syst., vol.31, no.3, pp.187–228, 2006.
[3] T. Dalamagas, T. Cheng, K.J. Winkel, and T. Sellis, “Clustering XML documents using structural summaries,” EDBT Workshop on Clustering Information over the Web (ClausWeb04), Heraklion, Greece, 2004.
[4] A. Nierman and H.V. Jagadish, “Evaluating structural similarity in XML documents,” Fifth International Workshop on the Web and Databases (WebDB 2002), 2002.
[5] S. Flesca, G. Manco, E. Masciari, L. Pontieri, and A. Pugliese, “Fast detection of XML structural similarity.” IEEE Trans. Knowl. Data Eng., vol.17, no.2, pp.160–175, Feb. 2004.
[6] W. Lian, D.-W.-L. Cheung, N. Mamoulis, and S.-M. Yiu, “An efficient and scalable algorithm for clustering XML documents by structure,” IEEE Trans. Knowl. Data Eng., vol.16, no.1, pp.82–96, Jan. 2004.
[7] M. Kozielski, “Improving the results and performance of clustering bit-encoded XML documents,” Sixth IEEE International Conference on Data Mining-Workshops (ICDMW’06), 2006.
[8] J. Yuan, X. Li, and L. Ma, “An improved XML document clustering using path feature,” FSKD, Fifth International Conference on Fuzzy Systems and Knowledge Discovery, vol.2, pp.400–404, 2008.
[9] H. Leung, F. Chung, S.C.F. Chan, and R. Luk, “XML document clustering using common XPath,” Proc. International Workshop on Challenges in Web Information Retrieval and Integration, Tokyo, pp.91–96, April 2005.
[10] A. Termier, M-C. Rousset, and M. Sebag, “Treefinder: A first step towards XML data mining,” Proc. IEEE International Conference on Data Mining, Maebashi, pp.450–457, Dec. 2002.
[11] J. Yang, W.K. Cheung, and X. Chen, “Learning the kernel matrix for XML document clustering,” Proc. 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (IEEE’05), pp.353–358, Hong Kong, April 2005.
[12] J. Liu, J.T.L. Wang, W. Hsu, and K.G. Herbert, “XML clustering by principal component analysis,” Proc. 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’04), pp.658–662, Boca Raton, Nov. 2004.
[13] J.W. Lee, K. Lee, and W. Kim, “Preparation for semantic-based XML mining,” 2001 IEEE International Conference on Data Mining, pp.345–352, San Jose, Nov. 2001.
[14] M.H. Qureshi and M.H. Samadzadeh, “Determining the complexity of XML documents,” Proc. International Conference on Information Technology: Coding and Computing (ITCC’05) - Volume II - Volume 02, pp.416–421, 2005.
[15] X.-Y. Li, “Using clustering technology to improve XML semantic search,” Proc. Seventh International Conference on Machine Learning and Cybernetics, vol.5, pp.2635–2639, July 2008.
[16] S. Zhang, J.T.L. Wang, and K.G. Herbert, “XML query by example,” Int. J. Comput. Intell. Appl., vol.2, no.3, pp.329–337, 2002.
[17] J. Robie and R. Hat, “XML Processing and Data Integration with XQuery,” IEEE Internet Computing, vol.11, no.4, pp.62–67, Aug. 2007.
[18] N. Bruno, N. Koudas, and D. Srivastava, “Holistic twig joins: Optimal XML pattern matching,” Proc. SIGMOD Conference, pp.310–321, 2002.
[19] Q. Zhao, L. Chen, S.S. Bhowmick, and S.K. Madria, “XML structural delta mining: Issues and challenges,” Data Knowl. Eng.,
Hsu-Kuang Chang received the B.S. degree in Computer Science from New York Institute of Technology in 1989 and the M.S. degree in Computer Science from New York Polytechnic University in 1991. He is a Ph.D. candidate in National Kaohsiung First University of Science and Technology, Taiwan. He is also a lecturer in the Department of Information Engineering, I-Shou University, Taiwan. His research interests include data mining, multimedia database, and information retrieval.