

Cost-Driven Storage Schema Selection for XML

Shihui Zheng

Fudan University
Shanghai, China
Shzheng0@Fudan.edu.cn

Ji-Rong Wen

Microsoft Research Asia
Beijing, China
Jrwen@Microsoft.com

Hongjun Lu

Hong Kong Univ. of Science & Technology
Hong Kong, China
luhj@cs.ust.hk

Abstract

Various models and approaches have been proposed for mapping XML data into relational tables recently. Most of those approaches produce relational schema for given XML data, based on pre-defined rules, heuristics, and user specifications, without considering workload. As the result, the schema obtained is often not optimal with respect to query performance. In this paper, we present a cost-driven approach to generate a near-optimal relational schema for a given XML data and expected workload, in the presence of space constraint. An efficient heuristic algorithm based on Hill Climbing is proposed together with a set of state transformation operations. Experimental study using the prototype system implementing the proposed algorithm, indicates that the produced schema can provide better performance than those well-known mapping approaches published in the literature.

1. Introduction

Anticipating that a large number of XML data will be available, various approaches to storing and querying XML data have been proposed, including storing XML data in semi-structured repository [6, 9], in object-oriented databases [5, 10], in relational systems [4, 7, 13], or building so-called native systems [15]. Because of the popularity of relational technology, storing and querying XML data over RDBMS became a practical approach pursuing by both researches and database vendors. One of the issues in all these processes of storing XML data in relational system is to create a relational schema for a given XML document so that the document can be stored based on the schema. The issue has been studied and various approaches have been proposed. Florescu and Kossmann compared alternative approaches of storing XML data as ternary relations [7]. Shanmugasundaram *et al.* proposed three approaches that derived relational schemas from Document Type Definitions (DTDs) of XML documents [13]. Deutsch *et al.* [4] proposed to

extract the regular structures of XML data, which in turn are transformed into relations by a pattern definition language. Those approaches use either mapping rules or user-defined scripts to determine the relational schema. However, pre-defined rules may not be able to produce for XML documents with arbitrary complexity a good relational schema that allows stored XML document to be efficiently accessed. On the other hand, it is impractical to have ordinary user to specify the mapping schema in many cases. In fact, even professional users or experienced database administrators will feel difficult to design a good schema for a complex XML document, especially if high query performance is required.

In this paper, we cast the problem of finding a good mapping from XML data to relational tables an optimization problem: Given an XML document together with its DTD, and a set of queries as the expected workload, find a relational schema that maps the XML document to relational tables, and minimizes the execution time (cost) of those queries. It can be shown that exhaustive search for the optimal mapping is impractical because of the number of feasible schema for an XML document, i.e., the search space, is usually very large. We developed an approach that uses Hill Climbing method to find a near optimal solution for the problem. Using this approach, a particular XML to relation mapping schema is viewed as a state. A set of operations is defined so that, given two such schemas, we can find a series of operations to transform one schema to the other one. On the other hand, the cost of executing the given workload can be estimated using a cost model. Starting from an initial schema, an optimal, or near optimal mapping schema can be found in reasonable time by searching for schema that reduces the cost.

While Hill Climbing is a classical solution to optimization problems, the work presented in this paper made the following contributions:

- Just like a complete database design process consists of conceptual design, logical design, and physical design, where physical design focuses on performance,

we strongly believe that design of XML data should also take performance into consideration. In this sense, our work represents the whole process of designing an XML database, and most previous work can only be viewed as part of the designing process.

- We studied the key issues of applying Hill Climbing search strategy to this schema selection problem, including initial schema selection, a set of operations that transfer one schema to another, and the cost model for XML queries with path expressions.
- The proposed approach has been implemented in a prototype system that can automatically select a good relational schema for given XML documents and workload. We conducted extensive experiments to study the effectiveness of our approach, including using published benchmark, XMark. The results indicated that the system is efficient and the obtained schema leads to better query performance than the schema produced using approaches published in the literature.

The remainder of the paper is organized as follows. Section 2 shows a motivating example and defines the problem of selecting a good relational mapping schema for XML data. Section 3 presents the framework of the proposed algorithm and its key components. Section 4 reports the experimental results. Section 5 briefly discusses related work and gives conclusions.

2. Preliminaries

In this section, we first illustrate our motivations with an example, and then give the problem statement for the problem of relational schema selection for XML.

2.1. Motivating Example

In this paper, we use a directed, labeled graph, *schema graph*, to model the structure of an XML document (represented by XML schema or DTD). In the schema graph, nodes represent elements, attributes or PCDATA. A special node, *, indicates 1:n relationships between the nodes at two ends of the * node. The formal definition of schema graph will be given later in Section 3. Figure 1 depicts a sample schema graph for XML data of an auction Web site defined in an XML benchmark, XMark [16] (for simplicity, we omit the PCDATA nodes here). Intuitively, such a schema graph can be mapped into relational schema arbitrary. For example, we can have a schema with the least number of relations, that is, to map an element and all of its descendants not truncated by * node to one relation¹. For the schema graph shown in Figure 1, we obtain four relations: *site*, *person*,

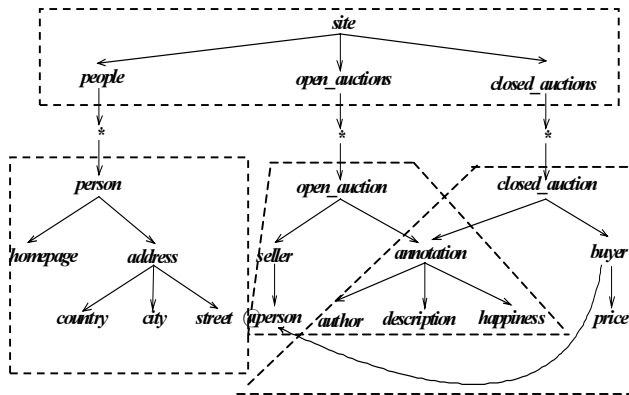


Figure 1. Schema graph for XML data of an auction Web site

open_auction, and *closed_auction*, as shown by the dashed lines. On the other extreme, we can have a schema with as many relations as possible, i.e. each element (attribute) in the schema graph is mapped to a separate relation, and we will have 20 relations for the schema graph in Figure 1. It is obvious that the first mapping schema has less element fragments, hence requires less joins when processing queries involving many elements, compared to the second mapping schema. However, such a schema suffers for other type of queries. Consider the following query: *list the names of persons and the number of items they bought*. It requires joining three relations: *site*, *person* and *closed_auction*, but only a few attributes in those relations are actually needed to answer the query. In such case, it could be better to have relations with smaller number of attributes to reduce the amount of data accessed. Since it is often the case that a workload consists of various types of queries, selecting a schema that can give overall optimal performance is not a trivial problem.

2.2. The Schema Selection Problem

Now we define the problem of selecting a relational schema for XML documents. First, we model a workload as

$$W = \{(Q_i, w_i), i = 1, 2, \dots, n\}$$

where Q_i is an XML query, e.g. an XQuery statement [3], w_i is its weight, reflecting its relative importance within the workload. A high weight could mean that the query is more frequently asked, or the query should be executed with less cost. With different mapping schema, the cost of the workload could be different. The cost of workload W against a particular mapping state, i.e. a schema S , is the weighted sum of the cost of the queries in the workload:

$$Cost(W, S) = \sum_{i=1}^n W_i * Cost(Q_i, S)$$

¹ RDBMS does not support attributes with set values, so those descendants with N:1 relationship have to map into separate relations.

$S=(f_1, f_2, f_3, f_4)$
 $f_1=(\text{site}, \text{site.people}, \text{site.open_auctions}, \text{site.closed_auctions})$
 $f_2=(\text{person}, \text{person.homepage}, \text{person.address}, \text{person.address.country}, \text{person.address.city}, \text{person.address.street})$
 $f_3=(\text{open_auction}, \text{open_auction.seller}, \text{open_auction.seller.@person}, \text{open_auction.annotation},$
 $\text{open_auction.annotation.author}, \text{open_auction.annotation.description}, \text{open_auction.annotation.happiness})$
 $f_4=(\text{closed_auction}, \text{closed_auction.annotation}, \text{open_auction.annotation.author}, \text{open_auction.annotation.description},$
 $\text{open_auction.annotation.happiness}, \text{closed_auction.buyer}, \text{closed_auction.buyer.@person}, \text{closed_auction.buyer.price})$

Figure 2. A mapping state

The XML mapping schema selection problem can then be defined as follows.

Definition 1: Given a set of XML documents, and a workload W , find the optimal relational schema S for those documents, such that $Cost(W, S)$ is minimal.

The problem can be reduced into the problem of optimal partition of graph. Garey [8] has shown that the problem of optimal partition of graph is NP-Complete. Thus, performing an exhausting search is completely impractical. In this paper, we propose to use the Hill Climbing strategy to solve the problem, so that only part of the search space is searched for a near optimal schema for an expected workload in reasonable time.

2.3. Notations

In this study, we use schema graph to model the structure of an XML document defined by DTD or XML schema.

Definition 2: A *schema graph* $G = (V, E, \sum_v, lab, id, r)$ is a directed, labelled graph, where

1. V is a finite set of nodes.
2. $E \subseteq V \times V$ is a set of edges.
3. $\sum_v = element \cup attribute \cup \{PCDATA\} \cup \{*\}$ is a finite set of symbols.
4. lab is a mapping $V \rightarrow \sum_v$, which assigns a label to each node in V , a node can be an element, if $lab(V) \in element$, an attribute, if $lab(V) \in attribute$, text content, if $lab(V) = PCDATA$, or star, if $lab(V) = *$.
5. id is a mapping that assigns a unique identifier for each node.
6. r is the unique root of the schema graph.

When mapping a schema graph into relational schema, we maintain annotations, i.e. the statistics on the data instance, with each node in the schema graph. A schema graph with annotations is called an *annotated schema graph*.

Definition 3: A directed, labeled graph $G=(V, E, \sum_v, lab, id, r)$ is a *annotated schema graph*, if

1. G is a schema graph.
2. Each node $v \in V$ has an annotation set $(R, size(v), type, AQ)$, where
 - R is the relation that v is mapped into.

- $size(v)$ is the number of instances of element v in the XML document.
- $type \in \{string, int\}$ is the storing type for the node v .
- AQ is the set of queries that access v , that is, $AQ(v) = \{Q \mid Q \text{ accesses node } v\}$.

In the following discussion, we may call an annotated schema graph just a schema graph, if there is no confusion. The set of instances of a schema graph G is a set of XML documents conforming to the schema graph, which is also referred as the *extent* of G , denoted as $ext(G)$, that is

$$ext(G) = \{D \mid D \text{ is an XML document conforming to } G\}$$

A connected subgraph of schema graph G is called a *fragment*, denoted as $f=(v_1, v_2, \dots, v_m)$, where v_i is a node in f . Each fragment has a unique root, denoted as r_f . The *extent* of a fragment is all of the document instance fragments conforming to the fragment.

We say two fragments f_1 and f_2 are *neighboring*, if $f_1 \cap f_2 = \phi$, and there is an edge (u, v) from f_1 to f_2 in G , where $u \in f_1, v \in f_2$. We say two fragment f_1 and f_2 are *twin*, if

- 1) $ext(f_1) \cap ext(f_2) = \phi$.
- 2) for each node $u \in f_1$, there is a node $v \in f_2$, s.t. $lab(v) = lab(u)$.
- 3) for each node $v \in f_2$, there is a node $u \in f_1$, s.t. $lab(u) = lab(v)$.

We call a partitioning scheme S on an annotated schema graph G a *mapping state*, hereafter referred as *state*, if $S=(f_1, f_2, \dots, f_n)$, where f_i is a fragment in G , s.t. $S = \bigcup_{i=1}^n f_i$, and for any two fragments f_i and f_j , if $i \neq j, f_i \cap f_j = \phi$ holds. As an example, Figure 2 shows such a state for the schema graph in Figure 1.

Note that a fragment $f=(v_1, v_2, \dots, v_m)$ in S can be mapped to a relation R in a straightforward fashion as follows: each node v_i becomes an attribute of R , and its position in G is preserved by adding one attribute, the identifier of the parent of the root. That is:

$$R(id(r_f):integer, id(r_f.parent):integer, lab(v_1):type(v_1), lab(v_2):type(v_2), \dots, lab(v_m):type(v_m)).$$

where r_f is the root of f and $r_f.parent$ represents the parent of r_f (it is null for the root of a document). $lab(v_i)$ is the field name of v_i in R , $type(v_i)$ is the type for the field. For example, the relations corresponding to fragment f_1 and f_2 in Figure 2 are the *Site* and *Person* fragments in Figure 1.

It is easy to see that a state is one solution to the schema selection problem. All of the possible partitioning schemes on the schema graph consist of the solution space. Thus, the goal of schema selection is to find the optimal (or near optimal) solution in the state space, which makes the cost of the workload minimal.

3. Hill Climbing Algorithm for Schema Selection

Hill Climbing is one strategy searching for solutions for optimization problem that scales well. It can find the local optimal solution in a search space within a reasonable time. In this section, we first describe a Hill Climbing algorithm for the XML schema selection problem. After that, we discuss three key components of the algorithm: initial state selection, state transformation and estimation of the cost of workload.

Figure 3 outlines a Hill Climbing algorithm, HC, for our schema selection problem. The algorithm takes an initial mapping state S_0 , and a workload W as input. Starting with the initial state as the current state, the algorithm finds a neighboring state that can be reached from the current state through state transformation operations (discussed in section 3.4). If the cost of executing the workload at the new state is smaller than that at the current state, the new state is used as the current state. Otherwise, another neighboring state is selected and checked. This process is repeated until there are no more unvisited neighboring states. The final current state is returned as the optimal partitioning schema. Three key components of the algorithm are:

- 1) selection of the initial state;
- 2) transformation(s) from a state to its neighboring state(s);
- 3) estimation of the cost of the workload.

We will discuss those three parts in the following subsections in details.

3.1. Selecting an Initial State

It has been known that the initial state to a Hill Climbing search algorithm may affect both the efficiency of the search algorithm and the optimality of the search results. First, a good initial state may reduce the search time. That is, if the selected initial state is close to the optimal solution state, the time need to reach the optimal solution state can be expected to be shorter [11, 14]. Second, some initial states may lead the algorithm to local minima instead of the global minima. The degree of those effects varies depending on the property of the optimization problem itself, state transformation operations, as well as the effectiveness of the cost functions. We use the general practice: selecting an initial state using existing algorithms that give reasonable good

Algorithm HC

Input: An initial state $S_0 = \{f_1, f_2, \dots, f_n\}$, and a workload $W = ((Q_1, w_1), (Q_2, w_2), \dots, (Q_m, w_m))$;

Output: a state with minimal cost.

1. $CurS = S_0$;
2. **repeat**
3. $S =$ a unvisited neighbouring state of $CurS$;
4. **if** $S \neq \phi$ **then**
5. **if** $Cost(W, S) < Cost(W, CurS)$ **then**
6. $CurS = S$;
7. **until** $S = \phi$;
8. **return**($CurS$);

Figure 3. A Hill Climbing Algorithm

solutions. Particularly, in this study, we use three types of relational mapping for XML: *Attribute* mapping proposed in [7], and *Shared* and *Hybrid* mapping, proposed by in [13], to generate the initial state, respectively, and compare their effects on the algorithm efficiency.

3.2. State Transformation

The second component in the algorithm is state transformation. Recall that a state consists of fragments of a schema graph. The basic operations to reorganize fragments are cut, further partitioning a fragment into two, and merge, combining two fragments into one. Since both cut and merge can be done vertically and horizontally, we define four primitive operations. They are *V-Cut*, *V-Merge*, *H-Cut*, and *H-Merge*, defined as follows:

- *V-Cut*($S, f, (u, v)$): cut fragment f into two neighbouring fragment f_1 and f_2 , s.t. $f_1 \cup f_2 = f$, where (u, v) is an edge from f_1 to f_2 , i.e. $u \in f_1, v \in f_2$.
- *V-Merge*(S, f_1, f_2): merge f_1 and f_2 into one fragment $f = f_1 \cup f_2$.
- *H-Cut*($S, f, (u, v)$): split f into two twin fragments f_1 and f_2 horizontally from edge (u, v) , where $u \notin f, v \in f$, s.t. $ext(f_1) \cup ext(f_2) = ext(f)$, and $ext(f_1) \cap ext(f_2) = \phi$.
- *H-Merge*(S, f_1, f_2): merge two twin fragments f_1 and f_2 into one fragment f , s.t. $ext(f) = ext(f_1) \cup ext(f_2)$.

The *V-Cut* operation cuts a fragment into two neighboring fragments vertically along a path. While *H-Cut* splits the instance of a fragment into two disjoint groups, each of them becomes the instances of the two resulting fragments, respectively. In particular, we only apply *H-Cut* operation on the fragments that have more than one parent in schema graph and the shared fragments are horizontally split into two twin fragments with different parent set. Conversely, *V-Merge* and *H-Merge* operations merge two neighboring fragments and twin fragments respectively. It is easy to see that the *Cut* and *Merge* operations are symmetric. That is, given a *V-*

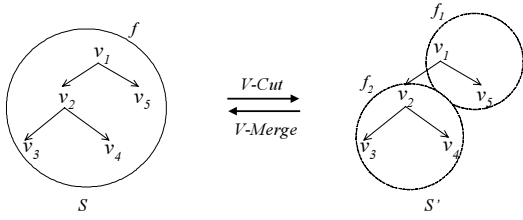


Figure 4. The V-Cut and V-Merge operations

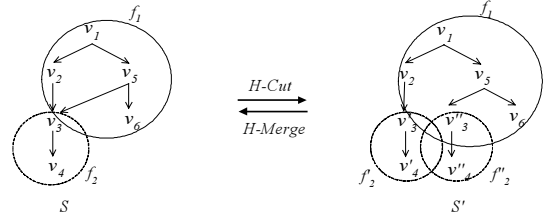


Figure 5. The H-Cut and H-Merge operations

Cut operation that cuts f in state S into two fragments f_1 and f_2 , reaching state S' , there exists a *V-Merge* operation merging f_1 and f_2 (in S') into f (in S), and vice versa. Similarly, given a *H-Cut* operation that cuts f in S into two fragments f_1 and f_2 , reaching state S' , there exists a *H-Merge* operation merging f_1 and f_2 (in S') into f (in S), and vice versa.

Example 1: Figure 4 and 5 illustrate those four operations by example. In Figure 4, a state S consists of only a fragment $f=(v_1, v_2, v_3, v_4, v_5)$. The *V-Cut* operation depicted in the figure cuts it into two fragments $f_1=(v_1, v_5)$ and $f_2=(v_2, v_3, v_4)$. The new state is S' . Conversely, the *V-Merge* operation merges the two fragment f_1 and f_2 into fragment f . Figure 5 shows a state consisting of two fragment f_1 and f_2 , and f_2 is cut into two twin fragments $f_2'=(v_3', v_4')$ and $f_2''=(v_3'', v_4'')$ by the *H-Cut* shown in the figure and the *H-Merge* operation merges the two fragments f_2' and f_2'' into fragment f_2 as well.

If there is a transformation T transforming S into S' , we say two states S and S' are *neighboring*. For example, both the state S and S' in Figure 4 and Figure 5 are *neighboring*. We say two state S and S' are *reachable*, if there is a sequence of primitive operations T_1, T_2, \dots, T_n transforming S into S' . It can be proved that, for any two states S and S' , there always exists a sequence of operations transforming S into S' . Thus, the state space is strong connected and the optimal state is *reachable* from any initial state.

Table 1. The parameters for cost model

Parameters	Description
$ E_i $	The number of instances of element E_i
C_i	The field width of element E_i
$Fan-out(i, j)$	The average fan-out from E_i to E_j
$ D_i $	The number of elements in the extend of f_i
$ f_i $	The size of the extend of fragment f_i
$Sel(l_1 / l_2 / \dots / l_n)$	The selectivity of simple path $l_1 / l_2 / \dots / l_n$
Sel_i	The selectivity of the path from root to f_i

3.3. Cost Model

In this section, we present a model to estimate the cost of the workload, which is used to compare two states given a mapping schema with annotations. We assume that queries in the workload are expressed using XQuery language [3]. Since an XQuery with regular path expressions can be translated into a group of queries with simple path expressions only, exploiting the schema [13], we only consider the cost for evaluating simple path expressions appearing in an XQuery in this model.

In Table 1, we listed the parameters used in this model together with their descriptions. Among those parameters, we assume that the number of instances of each element, $|E_i|$, and the length of each instance, C_i , are known from the annotations of each node in given schema graph. The parameter, $Fan-out(i, j)$, is in fact the average number of instances in E_j , a sub-element of E_i . It is collected from XML data at the beginning of the Hill Climbing algorithm. $|D_i|$ and $|f_i|$ are computed as following

$$|D_i| = \sum_{E_j \in f_i} |E_j| \quad \text{and} \quad |f_i| = \sum_{E_j \in f_i} |E_j| * C_j$$

We take the *Markov table* [1] to estimate the selectivity of a simple path from the selectivity of simple path with length 2, which is just the *Fan-out* between two elements. That is

$$Sel(l_1 / l_2 / \dots / l_n) = \prod_{i=1}^{n-1} Fan-out(l_i, l_{i+1})$$

It can be seen that the number of instances of elements is the base of estimation of other parameters. When horizontal transformation operations, *H-Cut* and *H-Merge*, are applied, number of instances will change in different fragments. The example in Figure 6 illustrates the issue. In Figure 6(a) and (b), the number on the right side of each node indicates its number of instances. Assume that fragment $f=(v_3, v_4)$ in Figure 6(a) is cut into two twin fragments $f_2'=(v_3', v_4')$ and $f_2''=(v_3'', v_4'')$ by *H-Cut*($S, f_2, (v_2, v_3)$). After the operation, v_3' represents all the target nodes of path $v_1/v_2/v_3'$, while v_3'' indicates all the target nodes of path $v_1/v_3/v_3''$. Similar situations happen on all of the nodes in f_2' and f_2'' . The exact number of instances of each element after the operation can be only obtained from scanning the original document, which is very expensive for large documents. Without scanning the original data, we estimate the sizes as follows. Assume operation *H-Cut*

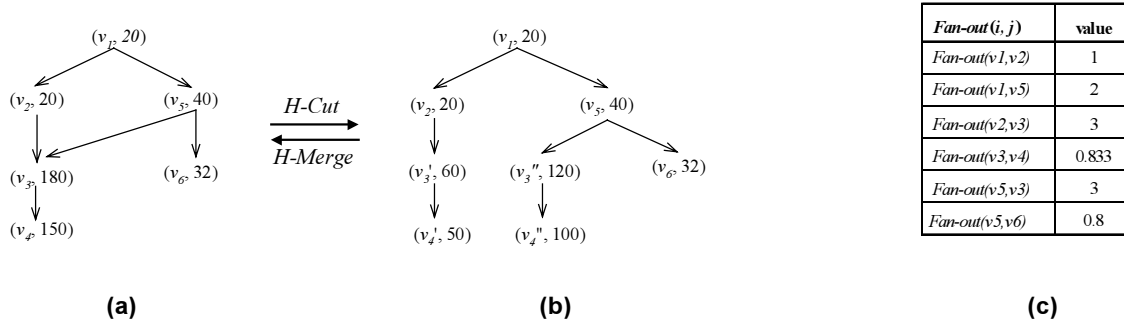


Figure 6. Node size estimation in state transformation

$(S, f, (u, v))$ splits $f=(v_1, v_2, \dots, v_m)$ into $f_1=(v_1', v_2', \dots, v_m')$ and $f_2=(v_1'', v_2'', \dots, v_m'')$, then the size of v_i' and v_i'' can be estimated as

$$|v_i'| = |u| * sel(P(u, v_i')) \text{ and } |v_i''| = |v| - |v_i'|$$

where $P(u, v_i')$ is the path from u to v_i' . Obviously, when two nodes v_i' and v_i'' are merged into node v , the size of node v in the result state is the sum of the sizes of v_i' and v_i'' .

Return to our example in Figure 6. Given the *Fan-out* values shown in the table of Figure 6(c), we can derive that the sizes of v_3' and v_3'' are 60 and 120, respectively, and the sizes of the other nodes in the new state can also be derived as well.

Now, we give the cost formula for an XQuery query, Q_i . Let the fragment set accessed by query Q_i be $\{f_{i1}, f_{i2}, \dots, f_{ik}\}$. We model the cost of executing query Q_i as

$$Cost(Q_i, S) = \begin{cases} |f_{i1}| & \text{if } k=1 \\ \sum_{i,j} (|f_i| * Sel_i + \delta * (|D_i| + |D_j|) / 2) & \text{if } k > 1 \end{cases}$$

where f_i and f_j ($1 \leq i, j \leq k$, and $i \neq j$) are two join fragments, i.e. there are join conditions on the relations corresponding to those two fragments in Q_i . The rationale is as follows: if Q_i only accesses a fragment, i.e. $k=1$, then the cost of Q_i is equal to the size of the fragment. Otherwise, the cost is estimated by the join cost for evaluating the simple path expressions appearing in Q_i . Here we use the formula $|f_i| * Sel_i + \delta * (|D_i| + |D_j|) / 2$ to simulate the cost for joining the two relations corresponding to fragment f_i and f_j . δ is a coefficient. In our experiments, we set $\delta=3$ to simulate the cost of hash-based join. The cost of a workload W on a state S is therefore estimated as

$$Cost(W, S) = \sum_{i=1}^n w_i * Cost(Q_i, S)$$

4. Experiments

We have implemented the schema selection algorithm on Microsoft SQL SERVER 2000. We conducted a sequence of experiments to evaluate the effectiveness of

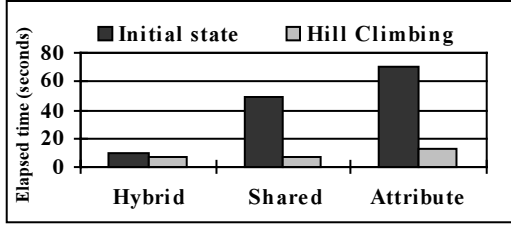
the tool. First, we examined the performance of the proposed algorithms starting from different initial states. Second, we compared the result of our algorithm with previous schema mapping approaches. Finally, we studied the robustness of our algorithm with workloads represented by different weight assignments of queries.

4.1. Experimental Setup

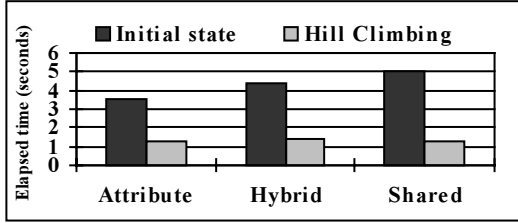
The experiments were conducted on a machine with 2G CPU and 1G RAM. The test data and workload used for our tests are from the XML benchmark *XMark* [16]. The test data with size 113M are generated with standard scaling factor 1. In our study, we formed two workloads from *XMark* queries [16], referred as Workload-1 and Workload-2 respectively. Workload-1 is composed of 16 complex queries. Workload-2, however, contains 8 “path expression-based” queries, whose cost is well captured by our cost model. In the first two experiments, each query is assigned with unit weight (set as 1). Those queries are translated into SQL statements following the approaches presented in [13]. In experiments, we use the actual executing time of workloads, as well as the estimated cost, as the measure of the quality of results schema.

4.2. Evaluating the Impact of Initial States

This set of experiments examines the impact of the initial states on the quality of the recommended storage schema produced by the Hill Climbing algorithm. We take the previous proposals *Attribute* [7], *Shared* and *Hybrid* [13] mapping as initial states, respectively, obtaining three variations of the Hill Climbing algorithm, referred as *Hybrid-HC*, *Shared-HC* and *Attribute-HC* respectively. Figure 7 (a) and (b) shows the performance of the different initial mappings and that of final mappings generated by Hill Climbing starting from those initial states on Workload-1 and Workload-2, respectively. We can see that, on Workload-1, Hill Climbing obtains the similar performance starting from *Shared* and *Hybrid* mapping, both outperforming that starting from *Attribute*



(a)



(b)

Figure 7. Overall cost (executing time) on Workload-1 (a) and Workload-2 (b)

mapping. On this workload, *Hybrid* mapping results a rather good storage schema, while the cost of *Attribute* mapping is extremely expensive. However, the result schema generated by *Attribute-HC* algorithm improves the performance of *Attribute* dramatically. On Workload-2, *Attribute* mapping is a better starting point than *Shared* and *Hybrid* mapping. However, Hill Climbing reaches the similar (sub) optimal solutions from the three initial states.

Figure 8 depicts the estimated cost of the mapping schemas obtained from those three algorithms on successive transformations for Workload-2. We can see that the estimated cost of *Hybrid-HC* drops faster than that of *Shared-HC* and *Attribute-HC*. *Shared-HC* and *Attribute-HC*, reach the optimal states after the same number of transformations and *Hybrid-HC* costs a little more (12 transformations). However, the final schemas generated by those three algorithms have the similar performance. It is consistent with the actual executing time of those queries in Workload-2.

The reason that our Hill Climbing algorithm for schema selection is not sensitive to the initial state can be explained as following. Unlike the classical Hill Climbing algorithm suffering from the limitation in choosing the direction of the motion, our Hill Climbing algorithm allows diverse directions of motion, provided by the transformation operation set, which helps the Hill Climbing always find the ascent directions, not be stuck in local optimum, wherever initial point it starts from.

4.3. Performance of Hill Climbing Algorithm

Now, we move to the performance of our schema se-

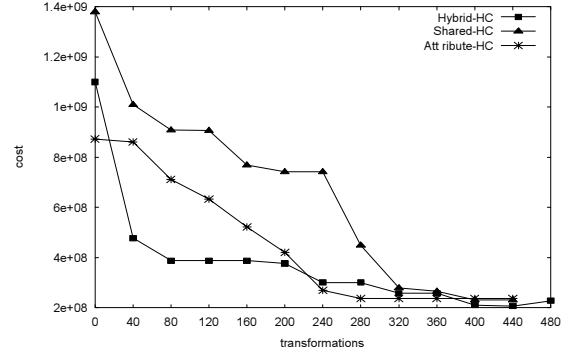
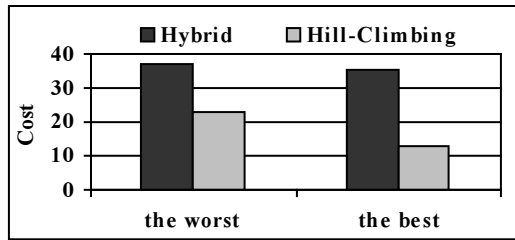


Figure 8. The impact of different initial states

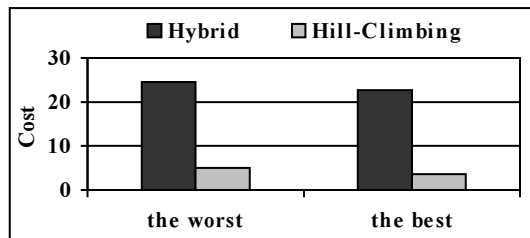
lection tool. From Figure 7 (a) and (b), we can see that the Hill Climbing can pick out much better storage mappings than the inlining or binary partition approaches for various queries. On Workload-1, Hill Climbing reduces the cost of *Hybrid* by 35% and *Attribute* by 82.7%. The cost of Workload-2 on the schema generated by Hill Climbing is only 24.9% of that generated by *Hybrid*, and 35.1% of that by *Attribute*. Hill Climbing shares the benefit of the inlining (*Hybrid* and *Shared*) approaches by vertical merge and also removes the columns that are not relative or less relative to the workload by vertical cuts. It clusters the elements that commonly accesses by queries together and thus obtain better performance. Moreover, horizontal cut operation removes the rows that are not shared by queries, thus reduces the cost for path traversal further.

4.4. Robustness of the Algorithms

In the third set of experiments, we study the robustness of our algorithm on various workloads with different weight assignments. *Hybrid* mapping is used as the initial state of the Hill Climbing algorithm in experiments. We use a simple “80/20” rule to select 20% of elements randomly that are *hot*, i.e. we assume those elements are most frequently accessed in a certain period. Queries that access those *hot* elements are in turn assigned 4 times of weights more than others. In experiments, we obtained a large number of weight assignments with a random generator and get the different storage configurations. We tried a large number of groups of weight assignments and picked out the best and the worst cases of those different weight assignments. Figure 9 (a) shows that, on Workload-1, the overall cost of the Hill Climbing is 36% and 61% of the *Hybrid* mapping in the best and the worst case, respectively. Note that the *Hybrid* mapping is already a very good storage strategy for Workload-1. The overall performance under the worst case is acceptable. Figure 9 (b) shows that on Workload-2, Hill Climbing is better than *Hybrid* 85% in the best and 80% in the worst



(a)



(b)

Figure 9. The best and worst cases of the overall cost on Workload-1 (a) and Workload-2 (b)

case, respectively. Thus, the performance of Hill Climbing is less affected by the different workloads.

5. Conclusions

Various approaches mapping XML data into relational databases have proposed [4, 7, 12, 13] recently. All of those mapping approaches may not be optimal for a certain type of queries and XML documents. The storage schemas obtained from those approaches, however, are contained in the search space of our Hill Climbing algorithm. Most recently, Philip Bohannon et al. proposed a cost-based greedy algorithm for storing XML in relational databases [2]. However, they only considered a restrictive set of transformations, which can be viewed as a special case of our vertical merge/cut operation. Moreover, they do not consider horizontal partition. A large number of valuable storage configurations may be missed in their greedy search. We proposed a complete, flexible transformation set, which allows more flexible transformations in search.

The future work of this paper includes: use the XQuery based cost optimizer of RDBMS to get the more accurate cost estimation for each mapping state; conduct comprehensive comparison with the cost-based approach proposed in [2]; and allow redundant storage of XML documents, etc.

References

[1] A. Aboulnaga, A. R. Alameldeen, J. F. Naughton.

Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. Proceedings of the International Conference on Very Large Data Bases (VLDB), Roma, Italy, Sept. 2001. page 591-600.

[2] P. Bohannon, J. Freire, P. Roy, J. Siméon. From XML Schema to Relations: A Cost-based Approach to XML Storage. Proceedings of ICDE 2002.

[3] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: A query Language for XML. Technical report. World Wide Web Consortium, Feb. 2001. Available: <http://www.w3.org/TR/xquery>.

[4] A. Deutsch, M. Fernandez, D. Suciu. Storing Semistructured Data with STORED. Proceedings of SIGMOD, 1999.

[5] eXeclon: the XML application development environment. <http://www.odi.com/execlon/main.htm>.

[6] M. F. Fernandez, D. Florescu, J. Kang, and et al. Catching the Boat with Strudel: Experiences with a Web-Site Management System. Proceedings of SIGMOD, 1998, 414-425.

[7] D. Florescu, D. Kossmann. A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database. Technical Report 3684, INRIA, March 1999.

[8] M. R. Garey, D. S. Johnson, L. J. Stockmeyer. Some Simplified NP-Complete Graph Problems. Theoretical Computer Science. 1(3): 237-267 (1976)

[9] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999.

[10] C. Kanne, G. Moerkotte, Efficient Storage of XML Data. Proceedings of the 16th International Conference on Data Engineering, 2000, San Diego, California, page 198.

[11] K. Park. A comparative study of genetic search. In Proc. 6th International Conference on Genetic Algorithms, pp. 512-519, July, 1995.

[12] A. Schmidt, M. L. Kersten, M. Windhouwer, F. Waas. Efficient Relational Storage and Retrieval of XML Documents. WebDB (Informal Proceedings) 2000: 47-52.

[13] J. Shanmugasundaram, K. Tufte, C. Zhang, and et al. Relational Databases for Querying XML Documents: Limitations and Opportunities. Proc. of the 25th International Conference on Very Large Databases, Edinburgh, Scotland, September 1999.

[14] M. Srinivas and L. M. Patnaik. Genetic Algorithms: A Survey. Computer, June, 1994, 17-26.

[15] Tamino: the infoamtion server for electric business. <http://www.softwareag.com/tamino>

[16] XMark: the XML-benchmark Project. <http://monetdb.cwi.nl/xml/>