Storing XML (with XSD) in SQL Databases: Interplay of Logical and Physical Designs

Surajit Chaudhuri, *Member*, *IEEE*, Zhiyuan Chen, *Member*, *IEEE*, Kyuseok Shim, *Member*, *IEEE*, and Yuqing Wu, *Member*, *IEEE*

Abstract—Much of business XML data has accompanying XSD specifications. In many scenarios, "shredding" such XML data into a relational storage is a popular paradigm. Optimizing evaluation of XPath queries over such XML data requires paying careful attention to both the logical and physical designs of the relational database where XML data is shredded. None of the existing solutions has taken into account physical design of the generated relational database. In this paper, we study the interplay of logical and physical design and conclude that 1) solving them independently leads to suboptimal performance and 2) there is substantial overlap between logical and physical designs: some well-known logical design transformations generate the same mappings as physical design. Furthermore, existing search algorithms are inefficient to search the extremely large space of logical and physical design combinations. We propose a search algorithm that carefully avoids searching duplicated mappings and utilizes the workload information to further prune the search space. Experimental results confirm the effectiveness of our approach.

Index Terms—XML, physical design, relational databases.

1 Introduction

XML has become the standard for exchanging and querying information over the Web. Furthermore, much of business XML data increasingly relies on accompanying XSD schema specifications [1] to ensure meaningful exchanges of information. Languages such as XPath (http://www.w3.org/TR/xpath) and XQuery have been proposed for querying XML data. One approach toward supporting XPath[1] over such XML data is building a native XML repository [2]. Alternatively, in many scenarios, "shredding" such XML data (with its associated XSD specification) into a relational database is an attractive alternative for storage as it can take the full advantage of mature relational database technology [8], [4]. The latter approach requires accomplishing the following two tasks to ensure efficient execution of XPath queries over XML data: 1) design the *logical* mapping from XML schema to relational schema and 2) select *physical design structures* (i.e., indexes, materialized views, and partitioning) of the relational database where XML is shredded.

There has been much work on the logical design step [5], [9], [10], [12], [13], [15], [16], [18], [19], [20], [22]. However, past work has ignored the role of physical design at the relational level in the above optimization steps. This has two important ramifications. First, solving logical and physical designs independently leads to suboptimal query performance. Second, taking physical design into account in fact influences the definition of the appropriate search space for logical designs as well as how this space can be effectively searched. We begin by demonstrating each of these issues.

1. In this paper, we focus on XPath since, at this time, it is the most widely used subset of query languages.

- S. Chaudhuri is with the Microsoft Research, One Microsoft Way, Redmond, WA 98052. E-mail: surajitc@microsoft.com.
- Z. Chen is with the Information Systems Department, University of Maryland, Baltimore County, Baltimore, MD 21250. E-mail: zhchen@umbc.edu.
- K. Shim is with the School of Electrical Engineering and Computer Science, Seoul National University, Korea. E-mail: shim@ee.snu.ac.kr.
- Y. Wu is with the School of Informatics, Indiana University, Bloomington, IN 47405. E-mail: yuqwu@indiana.edu.

Manuscript received 22 Dec. 2003; revised 30 Nov. 2004; accepted 8 June 2005; published online 19 Oct. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0265-1203.

1.1 Suboptimal Performance

We use DBLP (http://dblp.uni-trier.de/xml) data as an example. DBLP stores publications in conference proceedings with the following subelements: `booktitle`, `title`, `year`, several `authors`, and `pages`. The schema of the data is shown in Fig. 1a. Using the hybrid-inlining mapping proposed in [20], publications are mapped to the following relations:

```
Mapping 1
 inproc(ID, PID, title, booktitle, year,
        pages)
 inproc_author(ID, PID, author)
```

The `PID` column in `inproc_author` table is a foreign key column that references the `ID` column in `inproc` table. An XPath query that returns the `title`, `year`, and `author` of SIGMOD papers can be translated to the following SQL statement using the sorted-outer union approach proposed in [21].

```
SELECT I.ID, title, year, NULL
FROM inproc I
WHERE booktitle = 'SIGMOD CONFERENCE'
UNION ALL
SELECT I.ID, NULL, NULL, author
FROM inproc I, inproc_author A
WHERE booktitle = 'SIGMOD CONFERENCE'
```

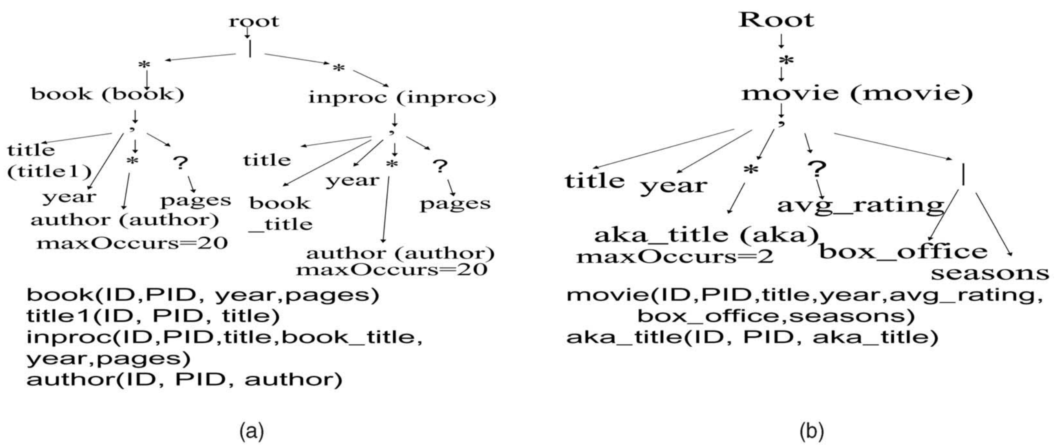


Fig. 1. XML schema for (a) DBLP and (b) Movie.

```
   AND I.ID = A.PID
ORDER BY I.ID
```

Since most publications have a small number of `authors`, we consider the following relational mapping where the first five `authors` of each publication are stored in the `inproc` relation (how we choose five will be discussed in Section 4.6), and the remaining `authors` are stored in the `inproc_author` relation.

```
Mapping 2
 inproc(ID, PID, title, booktitle, year,
        author_1, ..., author_5, pages)
 inproc_author(ID, PID, author)
```

Now, the SQL query becomes the following:

```
SELECT I.ID, title, year, author_1, ...,
 author_5, null
FROM inproc I
WHERE booktitle = 'SIGMOD CONFERENCE'
UNION ALL
SELECT I.ID, null, null, null, ..., null, author
FROM inproc I, inproc_author A
WHERE booktitle = 'SIGMOD CONFERENCE'
   AND I.ID = A.PID
ORDER BY I.ID
```

We ran both SQL queries on Microsoft SQL Server 2000 (the settings will be described in Section 5.1), both with the indexes and materialized views recommended by the Index Tuning Wizard of SQL Server [2], [7]. The data size is 100 MB and the space limit for both indexes, views, and data is set to 300 MB. The Tuning Wizard recommended several indexes for both mappings. The execution time using Mapping 2 was 0.25 second, and the time using Mapping 1 was 5.1 second. Under Mapping 2, the `inproc_author` relation is almost empty because most publications have no more than five authors. Thus, index-nested-loop join is used to join `inproc` and `inproc_author` under Mapping 2, and the join cost is much cheaper than the join cost under Mapping 1 where hash join is used. On the other hand, the `inproc` relation in Mapping 2 is larger than the `inproc` relation in Mapping 1, but a covering index[2] is recommended. Thus, the cost to retrieve `title` and `year` (the SQL before UNION ALL) is about the same under both mappings.

However, without indexes and materialized views, using Mapping 2 took 27 seconds and using Mapping 1 took 21 seconds because hash join was used under both mappings and it was more expensive to scan `inproc` under Mapping 2. Thus, if we first select the logical mapping without considering physical design, and then optimize the physical design of the selected mapping, we will wrongly select Mapping 1, leading to 20 times worse performance.

1.2 Definition of Logical Space

Previous work has been proposed several mapping transformations [5], [10], [18], [20] to represent different logical design options. However, in proposing the space of such mappings, the past work ignored the rich space of physical design. If the physical design space is not considered, then the logical design space gets inappropriately defined. As an illustration, consider the previous example. A mapping exploiting *outlining* transformation as proposed in [5], [10], [20] splits the `inproc` relation into two partitions, one consisting of columns referred in the query, the other consisting of the rest of other columns. This mapping achieves the same effect as a covering index under Mapping 1 and, thus, is in fact *subsumed* completely by physical database design. In this paper, we define that a logical transformation is `subsumed` by physical design if the performance benefits of applying the logical design transformation on a given XSD schema can be achieved by conducting physical design on the relational schema mapped from the original XSD schema.

Previous work [5], [18] did not consider the impact of physical design. Thus, a straightforward extension to their algorithm to search both logical and physical design will search outlining transformation along with physical design, which is inefficient. For example, an instance of such an extension (described in Section 4.2) took more than a day to

2. Defining covering index is a well known technique in relational databases to improve query performance. A covering index "covers" all the attributes of a table that are referenced in a query such that the query can be evaluated from the index only, without accessing the table.

optimize the storage of DBLP data for a rather small workload of queries. Such a tool is clearly not usable because real-life workloads often consist of thousands of queries and change over time, which requires reruning the tool from time to time.

This paper solves the combined problem of logical mapping from XML to relations and physical design of the corresponding database to avoid the above shortcomings. Specifically, the contributions of our paper are:

1. We find that:
   - Solving the logical mapping and the physical design problem independently leads to a suboptimal solution.
   - There is substantial overlap between logical and physical design: Many well-known logical design transformations are subsumed by physical design if they are applied alone. Therefore, the search space can be reduced significantly by pruning mappings generated by only applying subsumed transformations.
   - However, there also exists interactions between different transformations: The subsumed logical design transformations may be combined with other transformations to generate mappings not considered by physical design. However, we will show in Section 4 that considering such combinations does not introduce much overhead to the search algorithm.
2. We propose a *search algorithm* that judiciously explores the combined space of logical and physical design. The algorithm carefully avoids searching duplicated mappings and utilizes workload information to further prune the search space. Through experimental results, we demonstrate that the *quality* (in terms of the time to execute the query workload on resulting design) and *efficiency* (in terms of the search time) of our search algorithm is significantly better than existing known techniques.

We describe the previously proposed logical mapping options in Section 2 and study the impact of physical design on these mapping options in Section 3. We propose our solution in Section 4 and evaluate it experimentally in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2 Review of Logical Design Alternatives

In this section, we first give some preliminaries, then review the previously proposed logical mapping alternatives [5], [10], [18], [20], and finally define the problem of searching the combined space of logical and physical design.

We assume that an XML data has its XSD schema [25].[3] Following [18], we define a *schema tree* $T(V, E, A)$ to represent the XML schema.[4] V is a set of nodes representing the following type constructors: sequence (","), repetition ("*"), option ("?"), union (also called choice) ("|"), tagname, and simple type (corresponding to base types such as integer). The nodes may also have "minOccurs" and "maxOccurs" specifications in XSD. A set-valued element (i.e., maxOccurs $>$ 1) has its parent as '*' node, and an *optional node* (with zero or one occurrence, i.e., minOccurs = 0 and maxOccurs = 1) has a parent marked with "?." E are edges connecting nodes. A is a set of annotations for the nodes which specifies that a node will be mapped to a separate relation with the name as the annotation. Any node with an in-degree not equal to one (e.g., nodes without parents or with a '*' parent) must have an annotation because they must be mapped to a separate relation.

For example, Figs. 1a and 1b show part of the schema tree of DBLP and movie data sets. The annotations are specified in parentheses. Two types that are logically equivalent (see [18] for definition) but have distinct annotated parents are called shared type. For example, the two `title` elements in Fig. 1a are shared type.

Given a XSD tree $T(V, E, A)$, the mapping M from XML schema to relational schema $\mathcal{R}$, where $\mathcal{R}$ is a set of relations, is defined as follows:

1. Each node v with annotation in A is mapped to a relation with the annotation as table name. It has two default columns, an ID column as primary key that stores a unique node ID and a PID column as a foreign key that refers to the ID column in the table mapped from its parent.
2. Each leaf node l as descendants of v is mapped to a column in the table for v if there is no annotated node along the path between l and v.
3. If two nodes have the same annotation, they are mapped to the same table and the data instances for these two nodes are mapped to separate rows in the table.

2.1 Schema Transformations

A set of transformations on schema tree has been proposed in [5], [18] to define the space of logical mappings. We give a brief description below. Please refer to [5], [18] for details.

1. **Outlining and inlining.** Outlining introduces an annotation to a node v in the schema tree and v (along with its descendants) will be stored in a separate table. Inlining is the reverse. Outlining and inlining can group frequently coaccessed elements into a single table.
2. **Type Split/Merge.** Type split renames an annotation shared by multiple nodes. For example, `author` in the DBLP schema shown in Fig. 1 can be split into `book_author` and `inproc_author`. As a result, two tables will be created: one storing authors of `book` elements and the other storing authors of `inproc` elements. Type merge is the reverse. Type split and merge group frequently accessed occurrences of a shared type together.
3. **Union Distribution/Factorization.** Union distribution converts a schema fragment in form of $(a, (b|c))$ to $(a, b)|(a, c)$. For example, the movie element represented by `title,year, aka_title, avg_rating, (box_office | seasons)`, after union

3. Our work also applies to XML data with DTD by first transforming DTD to XSD.

4. The schema tree can also represents recursive types using shared type described below.

distribution and outlining, will be split into two tables: the MovieShow table which stores `title`, `year`, `aka_title`, `avg_rating`, `box_office`, and the TVShow table which stores `title`, `year`, `aka_title`, `avg_rating`, `seasons`. Union factorization is the reverse. c can also be empty (i.e., b is optional), which is called implicit union.

4. **Repetition Split/Merge.** Given a set-valued element E, repetition split converts E^* to k occurrences of E followed by E^*. It is followed by inlining the first k occurrences of E into the parent table. For example, in DBLP schema shown in Fig. 1, most books have no more than five authors, so we can store the first five authors of each book in the book table (by adding five columns, one for each inlined author). Repetition merge is the reverse. Repetition split can reduce the cost of join between parent and child tables. We have also limited repetition split to leaf nodes in the schema tree because in experiments we find that applying repetition split to intermediate nodes generates extremely complex join conditions, thus does not bring benefits.
5. **Associativity and Commutativity.** Associativity groups several types into one relational table. Commutativity changes order of types. Both of them can be combined with other transformations to generate new mappings.

Although we believe some of these transformations, such as union distribution/factorization, are applicable to recursive parts of schema as well. In this paper, we restrict ourselves to nonrecursive parts of XSD schema as in [5], [18].

We also consider a subset of XPath queries containing descendant and child axes because they are the most commonly used axes. For example, a query `//movie [title = "Titanic"]/(aka_title |avg_rating)` returns the average rating and aka_title of the movie with title "Titanic." `[title = "Titanic"]` identifies the selection condition and is called *selection path*. `aka_title` and `avg_rating` are results being returned and are called *projection elements*.

2.2 Problem Definition

Let W be an XPath workload that consists of a set of (Q_i, f_i), where Q_i is an XPath query and f_i is the weight associated with Q_i. Let a *configuration* F be a set of physical design structures on relational schema $\mathcal{R}$. Now, we define the problem of selecting a logical mapping along with its physical design.

Definition 1. *Given an XSD schema T, an XPath workload W, and a storage bound S, find a mapping M that maps T to a relational schema $\mathcal{R}$, and a configuration F on $\mathcal{R}$ whose storage requirement (including both data and physical design structures) does not exceed S, such that $\sum_{Q_i \in W} f_i \cdot cost(Q_i, \mathcal{R}, \mathcal{F})$ is minimized. $Cost(Q_i, \mathcal{R}, \mathcal{F})$ is the cost of running the SQL statements translated from Q_i on $\mathcal{R}$ with configuration F.*

We assume the presence of the XPath workload W as previous workload-based physical design [2], [7], [23] and logical design work [5]. In real life, the workload information may not be available. Thus, a practical solution is to start with some mapping (e.g., the hybrid inlining or shared inlining mapping proposed in [20]) and use our workload-based solution once the workload is known.

The combined search space of logical and physical design is extremely large because the number of possible combinations of physical design and logical design equals the number of possible physical design multiplied by the number of possible logical design, and the number of possible logical and physical design is large. For example, the number of indexes for a given set of relations is exponential to the number of columns [23]. The search space for logical design is also at least $\Omega(2^{|V'|})$, where V' is the set of nodes with in-degree of one because each such node can be either outlined or inlined. Thus, we need an efficient search algorithm over the space. In the next section, we study how physical design affects the space of logical design, which will help us prune the search space.

3 Impact of Physical Design on Search Space

In this section, we study the impact of physical design on the combined search space of logical and physical design. The physical design structures we considered are indexes, materialized views, and vertical partitioning. Specifically, we find that:

1. Outlining, inlining, associativity, and commutativity transformations alone are always subsumed by vertical partitioning, thus we call them *subsumed transformations*.
2. Other transformations bring extra benefits over physical design, thus we call them *nonsubsumed transformations*.
3. Subsumed transformations may be combined with nonsubsumed transformations to generate mappings that are not considered by physical design, thus the search space needs to include such combinations.

3.1 Subsumed Transformations

We define vertical partitioning of a set of relations $\mathcal{R}$ as $\mathcal{R}'$ such that 1) for each relation $R \in \mathcal{R}$, there exists one or more relations in $\mathcal{R}'$ such that the union of columns (except ID and PID columns) of latter relations equal the set of columns in R and 2) the latter relations share the same ID and PID columns.

We first prove that outlining, inlining, associativity, and commutativity transformations are subsumed by vertical partitioning.

Let $T(V, E, A)$ be the original XSD schema, C be a sequence of the above four types of transformations, and T' be the schema after applying C to T. Let T_0 be the XML schema transformed from T by inlining all nodes in T whose in-degree not equal to one.

Theorem 1. *The relations mapped from T' are a vertical partitioning of relations $\mathcal{R}_0$ mapped from T_0.*

Proof. The proof is straightforward. Any outlining, inlining, associativity, and commutativity transformation creates a vertical partitioning for a relation in $\mathcal{R}_0$ because it just partitions the columns in the same relation. By induction, any sequence of such transformations also creates a vertical partitioning of $\mathcal{R}_0$. Note that the vertical partitioning must be defined over the most inlined schema T_0 rather than T because, otherwise, the columns from different relations in T may be stored in the same relation by applying inlining. □

For example, for the movie schema in Fig. 1b, after applying commutativity on `year` and `avg_rating`, associativity on `title` and `avg_rating`, and outlining `title` and `avg_rating`, we create a vertical partitioning on `movie` table where one partition contains `title` and `avg_rating` and the other contains the remaining columns.

Since the above four subsumed transformations always generate a vertical partitioning of $\mathcal{R}_0$, these transformations do not generate any new relational schema compared to physical design on $\mathcal{R}_0$. Thus, if no transformations other than the above are considered, logical design is completely subsumed by physical design.

Note that if there is sufficient space for indexes and the query workload does not include update queries, vertical partitioning is also subsumed by covering indexes because both of them let database engine access frequently accessed columns instead of base tables [20].

3.2 Nonsubsumed Transformations

We now examine some examples of nonsubsumed transformations. Type split and merge change the horizontal layout of relations, so they are very similar to horizontal partitioning. However, in physical design, horizontal partitioning is based on either ranges or a random hash function on a set of columns, not on the XSD type level. Thus, type split and merge may generate relational schema not considered by horizontal partitioning.

Similarly, union distribution and factorization also generate relational schema not considered by horizontal partitioning because they use the "choice" group semantics in XSD to drop columns with all null values. For example, union distribution on the movie schema in Fig. 1b generates a movie table and a TV table where `box_office` columns is dropped in TV table because TV element does not have `box_office`.

Repetition split and merge are similar to a join view over the repetition element and its parent element. However, a join view needs to repeat the columns in the parent element for all repeated child element, while repetition split avoids such redundancy by storing such repeated occurrences in separate columns. This is possible because repetition split uses the "maxOccurs" semantics in XSD, which is not considered by join view.

Overall, the above transformations are not subsumed by traditional physical design because they use XSD specification such as "choice," "optional," and `maxOccurs` which imply complex constraints that are difficult to capture solely via physical design in relational databases.

3.3 Combination of Subsumed and Nonsubsumed Transformations

Although subsumed transformations alone do not generate a new relational schema, they may be combined with nonsubsumed transformations to generate a new relational schema. For example, type merge requires the two types being merged have exactly the same annotation. Thus, the two `title` elements in DBLP schema in Fig. 1a cannot be merged because they have different annotations (one with annotation "title1," the other with no annotation). However, merging the two titles becomes available if we first inline title of book (thus, the annotation "title1" will be deleted). Therefore, we need to consider mappings generated by combinations of subsumed and nonsubsumed transformations. There are two approaches of doing this: either consider such combinations in the logical design space and then conduct physical design, or add vertical partitioning to logical design space and delete it from physical design space. We adopted the first approach because we do not want to modify existing physical design tools.

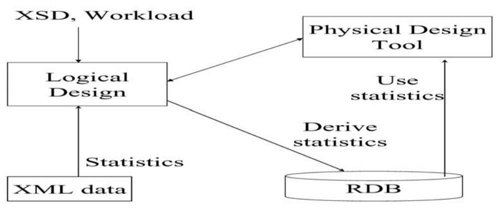


Fig. 2. Architecture.

4 Search Algorithm

We first describe the architecture of our logical and physical design tool (Section 4.1), then illustrate the problems of directly extending existing search algorithms for logical design (Section 4.2), and, finally, we describe our algorithm (Section 4.3 to Section 4.8).

4.1 Architecture

Fig. 2 illustrates the architecture of our system. It contains two key components: the logical design component that enumerates mappings and the physical design tool (the Index Tuning Wizard of Microsoft SQL Server 2000 in this paper) that optimizes the physical design of a mapping and returns the SQL query cost estimated by the query optimizer.

Following previous work on logical and physical design [2], [5], [7], [18], we use the query optimizer's cost model to estimate the query execution cost for each enumerated configuration.

The logical design component also needs to pass to the physical design tool the statistics on shredded relations and columns. We adopt the same approach in [13], [18] to collect statistics. The search always starts with a fully split schema where all possible outlining, union distribution, type split, and repetition split transformations are applied. Such a schema allows statistics to be collected on the finest granularity. Later, any generated schema can be transformed from the fully split schema by only using merge transformations (inlining, union factorization, type merge, and repetition merge). Thus, the statistics of such schema can be accurately derived from the statistics on the fully split schema. The reverse is not true because a split transformation such as union distribution generates difficulty for deriving statistics. For example, for the movie schema in Fig. 1b, suppose after applying union distribution, the `Movie` table is split into two tables: `TVShow` and `MovieShow`. It will be difficult to infer the statistics of elements shared by these two separated tables (e.g., `year, title`) because they have a finer granularity than the original `Movie` table. For example, the `year` values of `MovieShow` may be substantially earlier than the values in `TVShow`, which will be difficult to infer from the combined distribution of `year` in `Movie`.

The work in [13], [18] collects statistics directly from the XML data. For easiness of implementation, we collect statistics directly on the relational data loaded using the most split schema. The previous work collects the following three types of statistics:

1. the range of ID,
2. the distribution of PID, and
3. the value distribution for each column mapped from a base type.

Such statistics collected on relational data is exactly the same as those collected on XML data.

4.2 Drawbacks of Existing Solutions

A greedy algorithm has been previously proposed for logical design in [5], [18]. The algorithm iteratively enumerates mappings generated from the current mapping by applying transformations, and selects the mapping with the lowest cost to replace the current mapping and stops if there is no more cost reduction. We can certainly extend this algorithm to both logical and physical design by invoking the physical design tool instead of the query optimizer to estimate the cost of each mapping. However, such straightforward extension is not sufficient because the combined search space of logical and physical design is extremely large. The state-of-the-art physical design tool [2], [7] often requires hundreds of query optimizer calls and many database operations (such as creating tables and statistics). For each enumerated transformation on the XSD schema, the physical design tool needs to be called. We have implemented an instance of such extension. It needs to search 271 transformations on DBLP and tens of thousands of mappings (a mapping is generated by applying a sequence of transformations). Thus, it took more than a day to optimize a fairly small workload on DBLP which contains 10 queries. Since real-life workloads often consist of thousands of queries and change over time (requiring reruning the tool from time to time), reducing the tool's running time is crucial for the usability of the tool.

We have two key observations to improve the efficiency of search algorithm:

1. Avoid searching duplicated mappings. As illustrated in Section 3, we can reduce the number of mappings that need to be searched because many transformations are subsumed by physical design.
2. Prune search space using workload. Since the goal is to improve the performance of the given query workload, the workload itself can be used to prune the search space. For example, query `//movie[ title = "Titanic"]/ (aka_title | avg_rating)` likely benefits from repetition split on `aka_title`, which reduces the cost of join `movie` and `aka_title` relations, and union distribution on `avg_rating`, which enables the query engine to only access those movies with an `avg_rating`. All other transformations are unlikely to be beneficial to this query.

We next describe how to avoid searching subsumed configurations (Section 4.3) and how to use workload to prune the search space (Section 4.4 to 4.8).

4.3 Pruning of Subsumed Mappings

As described in Section 3.1, for a given XSD schema T, all relational schemas generated by subsumed transformations can be generated by vertical partitioning of the fully inlined schema T_0 transformed from T. Thus, our search algorithm only enumerates nonsubsumed transformations at each step. For each such transformation c, the schema after applying c is further fully inlined, and then the physical design tool is called to estimate the cost of that schema. This optimization is able to prune the search space significantly because the number of subsumed transformations is exponential in the XSD schema size. For example, the number of outlining/inlining transformations is $2^{|V'|}$, where $|V'|$ is the number of nodes whose in-degree does not equal to one.

However, as discussed in Section 3.3, a combination of subsumed and nonsubsumed transformations may generate mappings not subsumed by physical design. For example, merging the two title elements in Fig. 1a requires inlining one of them first. We adopt the *deep merge* solution proposed in [18] to solve this problem. The solution considers not only all directly applicable nonsubsumed transformations, it also considers all nonsubsumed transformations applicable after applying a sequence of either subsumed or nonsubsumed transformations. In the above example, type merging `title` becomes available after inlining the `title` element under `book`. It is important to note that identifying available nonsubsumed transformations does not require query cost estimation. Therefore, it does not add much overhead for search because the physical design tool needs not be called.

4.4 Workload-Based Pruning

The workload can be used to limit both the transformations we consider and to reduce the number of queries that need to be passed to physical design tool. We propose the following optimization techniques:

1. *Candidate selection*, which limits transformations by analyzing the query format of each individual query and only selects the transformations that will benefit that query.
2. *Candidate merging*, which avoids overfitting individual queries by generating candidate transformations that may improve the performance of multiple queries. Since there are many possible transformations generated by merging, we use a heuristic algorithm to further limit the number of merged transformations.
3. *Cost derivation*, which reduces the cost of calling physical design tool by deriving the cost of queries

```
Input: An XML schema $T$, an XML query workload $W$, a space limit $S$.
Output: The mapping $M_{min}$ and physical design $F_{min}$ with the minimal cost.
(01) $C_0$ = GenCandidate($T, W$);
     $C_1$ = MergeTypeCandidate($C_0$); $C_2$ = SplitTypeCandidate($C_0$)
(02) $M_0 = M_{min}$ =GenInitialMapping($T, C_2$)
(03) $C3$ = MergeCandidate($T, W, C_0$); $C = C1 \cup C3$
(04) $W_{SQL}$ = TranslateToSQL($T, M_0, W$)
(05) ($cost_{min}$, $F_{min}$) = PhysDesignTool($W_{SQL}, M_0, S$)
(06) while $C \neq \emptyset$ do
(07)     updated = false
(08)     for each $c \in C$ do
(09)         $M$ = ApplyTransformation($M_0, c$)
(10)         $W_{SQL}$ = TranslateToSQL($T, M, W$)
(11)         ($cost$, $F$) = PhysDesignTool($W_{SQL}, M, S$)
(12)         if $cost < cost_{min}$ then
(13)             ($cost_{min}$, $F_{min}$) = ($cost$, $F$)
(14)             $M_{min} = M$; $c_{min} = c$; updated = true
(15)         endif
(16)     endfor
(17)     if updated is false then break
(18)     else $M_0 = M_{min}$; $C = C - \{c_{min}\}$ endif
(19) endwhile
(20) return $M_{min}$, $F_{min}$
```

Fig. 3. Greedy Search algorithm.

using the new configuration from cost the obtained under previously enumerated configuration. This technique will reduce the number of queries need to be passed to the physical design tool. The physical design tool in turn takes less time to optimize a smaller workload instead.

Fig. 3 shows the sketch of the Greedy algorithm. For illustration, we omit the details of generating fully inlined mapping and deep merging in the algorithm. The algorithm first selects a set of candidate transformations at line 1. The merge type candidates are stored in C_1 and split type candidates are stored in C_2. It then generates an initial fully-split mapping M_0 at line 2 by applying all split type candidates. At line 3, candidates are merged. These newly generated candidates are added to C along with previously generated merge type candidates. At line 5, the algorithm calls the physical design tool to select physical design structures on M_0 using the SQL workload W_{SQL} translated from the XML query workload W at line 4. Lines 6 to 19 repeatedly select the minimal-cost mapping M_{min} that is transformed from the current mapping M_0 with a transformation in C.[5] In each round, the minimal cost mapping M_{min} is initialized as M_0. For each transformation $c \in C$, lines 8 to 16 enumerate a mapping M transformed from M_0 using c (line 9), and call the physical design tool to return the cost and physical configuration of M (lines 10 and 11). Lines 12 to 15 replace M_{min} with M if the cost of M is lower than the cost of M_{min}. At the end of the round, line 17 returns if no better mapping is found. Otherwise, line 18 replaces M_0 with M_{min}, deletes from C the transformation c_{min} that generates M_{min} and proceeds to the next round.

5. A sequence of subsumed transformations may be applied along with c.

The complexity of the algorithm is $O(|C|^2 P)$, where $|C|$ is the number of candidate transformations and P is the time for one call of the physical design tool. Next, we describe how to reduce $|C|$ by candidate selection and merging, and how to reduce P by cost-derivation.

4.5 Candidate Selection

We analyze every $Q \in W$ and select the transformations that may improve the performance of Q as follows:

1. Subsumed transformations are not selected because they are subsumed by vertical partitioning or indexes. Such transformations may still be combined with a nonsubsumed transformation, but they will not be applied alone.
2. Generate a union distribution/factorization or type split/merge if Q accesses less than half of the partitions generated by applying this transformation. For example, for the movie schema in Fig. 1b, if

the query only accesses box_office, a union distribution over (box_office | seasons) will be generated. In experiments, we find that if more than half of the partitions are accessed, then such a transformation usually does not bring substantial benefits.

3. Generate a repetition split for set-valued element v if Q refers v in projection or selection path, and v's maximal cardinality is less than a threshold c_{max} or more than x percent of v have cardinality less than c_{max}. These parameters can be tuned and we use $c_{max} = 5$ and $x = 80$ in our experiments. The next section will explain why we choose the repetition split count in such a way.

Note that all split type candidates will be applied at once to generate the initial mapping, and later only merge type candidates will be applied during the greedy search.

4.6 Selection of Repetition Split Count

The problem is how to choose the number of repeated elements to be inlined in parent table when the maxOccurs of a set-valued element is unspecified or large. Let k be the number to be inlined. A small k may not save much of the join cost, and a large k may introduce too many nulls in parent relation and blow up the space. We find that repetition split is still effective if the distribution of the cardinality is skewed to the low cardinality region, and under such cases a good k is the smallest k such that most instances of the element have cardinality smaller than k. For example, in the DBLP data, each publication could have up to 20 authors, but 99 percent publications have no more than five authors. For this specific data set, we find that splitting the first five authors achieves the best balance between performance and space. Note that here, we use the statistics of the cardinality of set-valued elements in addition to the maxOccurs specification to decide the count.

4.7 Candidate Merging

Candidate selection tries to optimize each query, but it may miss transformations that may benefit multiple queries.

For example, assume the year element is also optional in the Movie schema shown in Fig. 1. Consider the following two queries:

```
Q1: //movie/year
Q2: //movie/avg_rating.
```

$Q1$ returns year element of movies. $Q2$ returns avg_rating element of movies. Let c_1 and c_2 be implicit union distribution on year and avg_rating, respectively. c_1 creates two tables, one stores those movies with year element, and the other stores those movies without year element. $Q1$ only needs to access the first partition. However, those movies with avg_rating element may or may not have year element. Thus, Q2 must access both partitions, so c_1 does not improve the performance of $Q2$. Similarly, c_2 helps $Q2$ but not $Q1$.

Now, consider a transformation c_3 which splits the movie relation into two partitions: one storing the movies having either year or avg_rating, the other storing the rest of movies. Even though c_3 is not selected by candidate selection, it improves the performance of both queries. c_3 can be seen as the result of *merging*c_1 and c_2.

Let C_0 be the set of implicit union distribution candidates chosen after candidate selection. Since there may be $O(2^{|C_0|})$ possible merged transformations, we use a cost-based greedy algorithm to efficiently search useful merging. The algorithm first examines all pairs $c_i, c_j \in C_0$ that can be merged (i.e., they are on the same relation, and the set of optional nodes of c_i is not a subset of that of c_j or vice versa). Each such pair is merged into a new candidate c on the union of the optional nodes of c_1 and c_2. We estimate a benefit for c. The algorithm then chooses the merged pair c_{max} with the maximal benefit. c_{max} is output as a new candidate, and added to input C_0 to replace the original pair of candidates used to generate it. This process is repeated until no new candidate is added. Finally, since we only consider merge type candidates in search, each output candidate will be replaced with their union factorization counterparts. The complexity of the algorithm is $O(|C_0|^3)$ because in each round $|C_0|^2$ pairs are considered and there can be at most $|C_0|$ rounds.

For efficiency reasons, we use a heuristic model to estimate the benefit of a merged candidate. Note that we only use this model to prune out bad candidates and later use query optimizer cost model during search. In experiments, we also found that using the heuristic model does not degrade the quality of results (see Section 5.3.2 for details). The model only considers I/O savings. Let c_i be an implicit union on relation R. If a query Q accesses more than half of partitions generated by c_i, the benefit is assumed zero. Otherwise, let R_A be the partitions that Q accesses, $RS(Q)$ be the set of relations referred by Q, and $cost(Q)$ be the cost of Q under the current mapping (obtained by calling optimizer). The I/O saving of c_i over the query Q is

$$s(c_i, Q) = ((|R| - \sum_{Ri \in R_A} |Ri|) / \sum_{Rj \in RS(Q)} |Rj|) \cdot cost(Q),$$

where $|R|$ denotes the size of R. $\sum_{Ri \in R_A} |Ri|$ and $\sum_{Rj \in RS(Q)} |Rj|$ denote the total sizes of partitions of R and relations accessed by Q. The model assumes the cost is proportional to the total sizes of relations being accessed. We define the total saving of c_i over W as $\sum_{Q \in W} s(c_i, Q) f_Q$, which is the weighted sum of savings over each query. For example, consider the previous example. The benefit of merging implicit union on avg_rating and year for Q1 (or Q2) equals the size of movies having neither avg_rating nor year divided by the size of all movies times the cost of Q1 (or Q2). The benefit for both queries is positive and, thus, the merged candidate will be selected.

4.8 Cost Derivation

The physical design tool is called for every enumerated mapping and accounts for most of the running time. Hence, we consider how to reduce this time by deriving the cost of queries under a new mapping from the cost under previously enumerated mappings. The intuition is that a transformation only changes one or two relations, thus, it is likely that the cost of many queries in the workload are not affected.

Let $I(Q, M)$ be the set of relational objects (indexes, materialized views, relations, and partitions) that the query engine uses to answer query Q under mapping M, and $cost(Q, M)$ be the cost of running Q under mapping M. We assume that if $I(Q, M') = I(Q, M)$ for a new mapping M', $cost(Q, M') = cost(Q, M)$ because the same query plan will be used if the same set of relational objects is used to answer the query.

Suppose M' is transformed from M using transformation c, we use the following heuristic rules to decide when $I(Q, M') = I(Q, M)$.

- *Irrelevant relation rule:* let $RS(Q)$ be the set of relations referred by Q, if c does not change any relation in $RS(Q)$, then $I(Q, M') = I(Q, M)$.
- *Repetition split rule:* suppose $R \in RS(Q)$, and the plan under M accesses a covering index I on R but not the base relation R. If c is a repetition split over v, and v is not referred by the SQL statement translated from Q, then $I(Q, M') = I(Q, M)$.

The intuition of repetition-split rule can be explained by the following query

```
Q3://movie[year >= "1998"]/
   (title | box_office) .
```

Suppose we are considering a hybrid inlining mapping M_0 and a mapping M_1, where the first five `aka_titles` are stored in its parent table `movie`. If a covering index $I1$ on `year`, `title`, and `box_office` is recommended for both mappings, the index has the same size under both mappings because the `movie` relations in both mappings have the same number of tuples and the indexed columns are not affected by the repetition-split. Thus, $I(Q3, M_0) = I(Q3, M_1)$.

- *Union distribution/factorization and type split/merge rule:* if c is an (implicit) union distribution/factorization or type split/merge on $R \in RS(Q)$, $I(Q, M') = I(Q, M)$ for either of the following two cases: 1) Q refers all partitions generated (or merged) by c, but none of the partitions participates in joins and 2) a repetition split is applied on R in M.

The intuition of this rule can be explained by the following example. Consider query `//movie/title` and a mapping M_2 after applying union distribution on `box_office` and `seasons`. They satisfy the first condition of the union distribution rule because the query accesses both partitions of `movie` and there is no join. The query plan selects the `title` on both relations and concatenates the results. The cost of concatenation accounts for a small portion of total execution time because the results for each relation are already in memory and there is no need for duplicate removal.

The second condition of the rule is satisfied if a repetition split is applied on `movie`. Remember that repetition split is only applied when most movies will be stored in its parent table. Thus, the `movie` relation is almost empty. Hence, applying union distribution/factorization or type split/merge on `movie` does not affect the performance because the cost associated with `movie` relation is neglectable.

Now, we describe how to use cost derivation. Given a new mapping M', we assume there is a $W' \subseteq W$ such that for queries $Q_i \in W'$, a mapping M_i has been enumerated and $I(Q_i, M') = I(Q_i, M_i)$. We further assume the physical design tool returns the indexes and materialized views in $I(Q_i, M_i)$ and their sizes. We approximate the cost of queries under M' as follows: For all $Q_i \in W'$, we assume the physical design tool recommends exactly the same physical design structures in $I(Q_i, M_i)$. Thus, we have $cost(Q_i, M') = cost(Q_i, M_i)$, where $cost(Q_i, M_i)$ has been already computed. For all $Q_j \in W - W'$, we assume the physical design tool recommends exactly the same physical design structures as we call the tool with workload $W - W'$ and with a new space limit S' equals to old limit S subtracted by the sizes of indexes and materialized views in $\bigcup_{Q_i \in W'} I(Q_i, M_i)$.

For example, consider a workload consisting of Q3 above and the following query

```
Q4: //movie[year = "1997"]/
   (aka_title | title).
```

Suppose the space limit is set to 2 GB and we have enumerated a mapping M_0 shown in Fig. 1b and the tool has recommended a covering index $I1$ for $Q3$ with size 100 MB and two indexes $I2$ and $I3$ for $Q4$ with total size 200 MB. For new mapping M_1 where repetition split is applied on `aka_title`, $I(Q3, M_1) = I(Q3, M_0)$ by the repetition split rule. We reestimate $cost(Q4, M_1)$ by calling the Tuning Wizard with the space limit of 1.9 GB (2 GB minus size of $I1$) and workload $\{Q4\}$.

In practice, these heuristic rules may not hold. So, in the Greedy algorithm, we only use cost-derivation when we compute the costs of enumerated mappings (line 11 in Fig. 3), and reestimate the cost of the minimum cost mapping selected in each round (line 18) without cost-derivation.

5 Evaluation of Search Algorithms

This section experimentally evaluates various search algorithms. Our major findings are: 1) It is important to search logical and physical design together because the algorithm searching them independently leads to query performance on average a factor of two worse than the algorithms searching them together. 2) The optimization techniques (avoiding searching subsumed transformations and workload-based pruning) proposed in Section 4 drastically reduce the running time of the algorithm (on average by two orders of magnitude) with little or no degradation in terms of quality of results.

5.1 Setup

5.1.1 *Algorithms*

We implemented our *Greedy* algorithm in Fig. 3 with all the optimization techniques proposed in Section 4. Our program uses SQL Server 2000's index tuning adviser as a physical design tool.

We also implemented two other algorithms: 1) a *Naive-Greedy* algorithm as a straightforward extension of the greedy algorithm proposed in [5], [18], which calls the physical design tool for every enumerated logical mapping and 2) a *Two-Step* algorithm that first greedily selects the

TABLE 1
Characteristics of Data Used in Experiments

Characteristics	*DBLP*	*Movie*
Number of nodes in schema	228	32
Depth of XSD schema tree	3	8
Number of transformations in total	271	57
Number of non subsumed transformations	136	22
Number of unions & implicit unions	18	3
Number of repetitions	26	9
Number of shared types	22	2
Data size using default mapping	100 MB	1 GB
Space (including data and indexes) used by schema-based mappings	300 MB	2 GB

minimal cost logical mapping without considering physical design, then selects the physical design for the previously selected logical mapping. Both Naive-Greedy and Two-Step make no distinction between subsumed and nonsubsumed transformations, and do not use workload to prune the search space. The Two-Step algorithm also assumes a clustered index on primary key (i.e., the ID column) and a nonclustered index on PID column (used in the join with the parent relation) in its first phase. These two indexes represent the best guess of physical design without considering workloads or calling the Tuning Wizard.

5.1.2 Data Set

We used a synthetic data set `Movie` and a real data set `DBLP`. The characteristics of these two data sets and space limits are listed in Table 1. `Movie` data set simulates a movie site. The values in Movie data set follows uniform distribution. The `DBLP` [1] data set contains publication records. Both schemas are complex enough to apply all types of schema transformations listed in Section 2 (the number of such transformations and the number of unions, repetitions, and shared types are listed in Table 1). The space limit is determined such that there is enough space for all indexes recommended by the physical design tool.

5.1.3 Workload

The workloads are generated randomly by varying two parameters: the selectivity of selection conditions and the number of projections. We vary the selectivity (0.01-0.1 or 0.5-1) and the number of projections (1-4 or 5-20) in each workload. We expect split-based transformations (outlining, type split, union distribution, and repetition split) will benefit queries with smaller number of projections because such queries will access only some of the partitions generated by split transformations. On the other hand, we expect merge-based transformations (inlining, type merge, union factorization, repetition merge) will benefit queries with many projections because such queries will access most partitions.

Each workload consists of 20 queries. The number of joins in each workload depends on the logical design and varies from 0 to 8. The workloads are named after their characteristics. For example, a workload "HP-LS-20" consists of 20 queries with large number of projections and low selectivity. An example of such query on DBLP returns nine elements of papers published in the conference proceedings in the year 2000.

```
/dblp/inproceedings[year="2000"]/(title |
year | cdrom | cite | author | editor | pages |
booktitle | ee)
```

The following query belongs to "LP-HS-20."

```
/dblp/inproceedings/(title | author)
```

For DBLP data, because Naive-Greedy did not stop after running for five days, we also tested four smaller workloads each consisting of 10 queries. Thus, we have eight workloads for DBLP and four workloads for Movie.

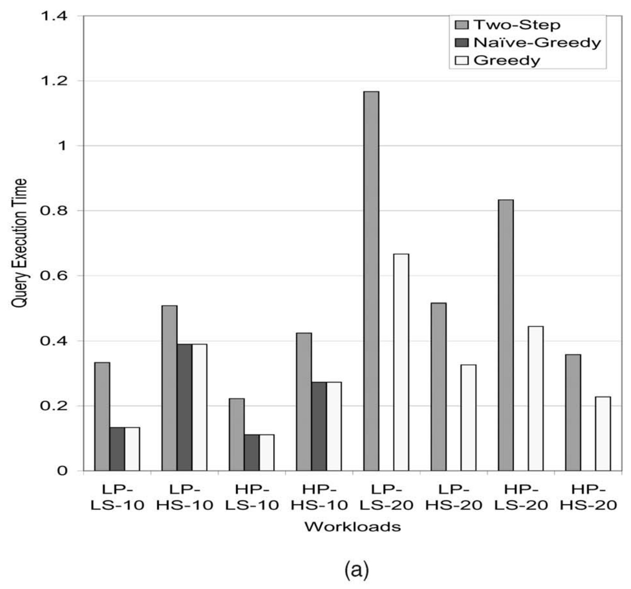
5.1.4 Quality Measure

We measured the quality of various algorithms by real execution time of queries on loaded relational databases with recommended indexes and materialized views. Since in practice, workload information may not be available, we compare the mappings returned based on workload information with the hybrid inlining mapping proposed in [20]. The hybrid inlining mapping is selected because it is not only one of the mappings with the best performance in [20], we also find in our experiments that it performs better than the fully split mapping when combined with physical design for the following reasons:

1. It inlines all subelements except those with in-degree not equal to one. Thus, it reduces the number of joins and join is expensive to compute.
2. Although hybrid inlining generates wide tables, physical design tool can recommend indexes, vertical partitions, and views on frequently accessed columns.

The execution time is normalized to the time under the hybrid inlining mapping with recommended indexes and materialized views.

We used a computer with two 2.4 GHZ CPU, 512 MB RAM, 40 GB single hard disk, and running Windows XP



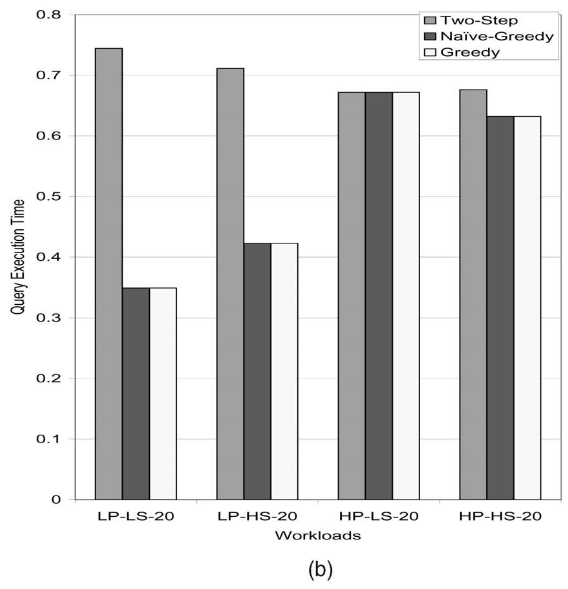


Fig. 4. Query execution time of mappings returned by Greedy, Naive-Greedy, and Two-Step, normalized to hybrid inlining mapping. (a) DBLB and (b) Movie.

Professional for our experiments. All queries were ran sequentially using SQL Server 2000. Each workload was ran in a cold cache, and the reported time was the average of three executions. The SQL Server does not allow users to set the memory size, but we observed that the memory used by different configurations for the same query was roughly the same.

5.2 Comparison of Various Algorithms

We first compare the quality of results returned by various algorithms. Figs. 4a and 4b report the query execution time of mappings returned by Two-Step, Naive-Greedy, and Greedy with recommended indexes and materialized views for DBLP and Movie data sets. We were unable to obtain results of Naive-Greedy for 20-query workloads on DBLP because it did not stop after five days. We have the following observations.

First, the results demonstrate the benefits of considering workloads. Greedy, Naive-Greedy, and Two-Step (except in one case) achieve significant improvement over the default hybrid inlining mapping.

Second, the results show the importance of considering interactions between logical and physical design: Two-Step algorithm leads to significantly worse performance than Greedy and Naive-Greedy (on average 77 percent worse for the DBLP data set and 47 percent worse for the Movie data set) and returns a mapping worse than the hybrid inlining mapping for one workload (LP-LS-20 for DBLP).

Finally, the results also demonstrate that Greedy and Naive-Greedy have almost the same quality for all workloads. Hence, the optimization techniques proposed in Section 4 lead to little loss of quality.

Fig. 5 reports the time of running Greedy, Two-Step, and Naive, normalized to the running time of Two-Step algorithm because it just does physical design once and is expected to be the fastest among the three algorithms.

Fig. 6 reports the number of transformations searched by Greedy and Naive (Two-Step searches the same set of transformations as Naive). Table 1 reports the number of all transformations and the number of nonsubsumed transformations.

The results demonstrate the clear advantage of avoiding searching subsumed transformations and using workload-based pruning techniques: Greedy searches about 10-40 times fewer transformations than Naive Greedy and Two Step for the DBLP data and about 5-10 times fewer transformations for the Movie data. The number of nonsubsumed transformations is also about a factor of two fewer than the total number of transformations.

As a result, Greedy is, on average, about two orders of magnitude faster than Naive-Greedy for DBLP and an order faster than Naive-Greedy for Movie (note the logarithm scale). The reason for lower speed-up for Movie is that Movie has a smaller schema than DBLP. Thus, the speed-up due to pruning transformations is lower. The number of transformations searched by Greedy also increases slightly as the workload size increases because there are more transformations that may benefit the additional queries in the workload.

Greedy's running time is comparable to Two-Step because Two-Step only searches physical design structures for a single logical mapping. However, Greedy returns configurations with much better query performance than Two-Step because Two-Step ignores interactions between logical and physical design. Thus, Greedy is the clear winner of the three algorithms.

5.3 Breakdown of Optimization Techniques

We next break down the effect of candidate selection, candidate merging, and cost derivation for the 20 query workloads on DBLP.

5.3.1 *Effectiveness of Candidate Selection*

Fig. 7 reports the speed-up of running time of Greedy due to avoiding searching subsumed transformations and the overall speed-up due to all candidate selection rules

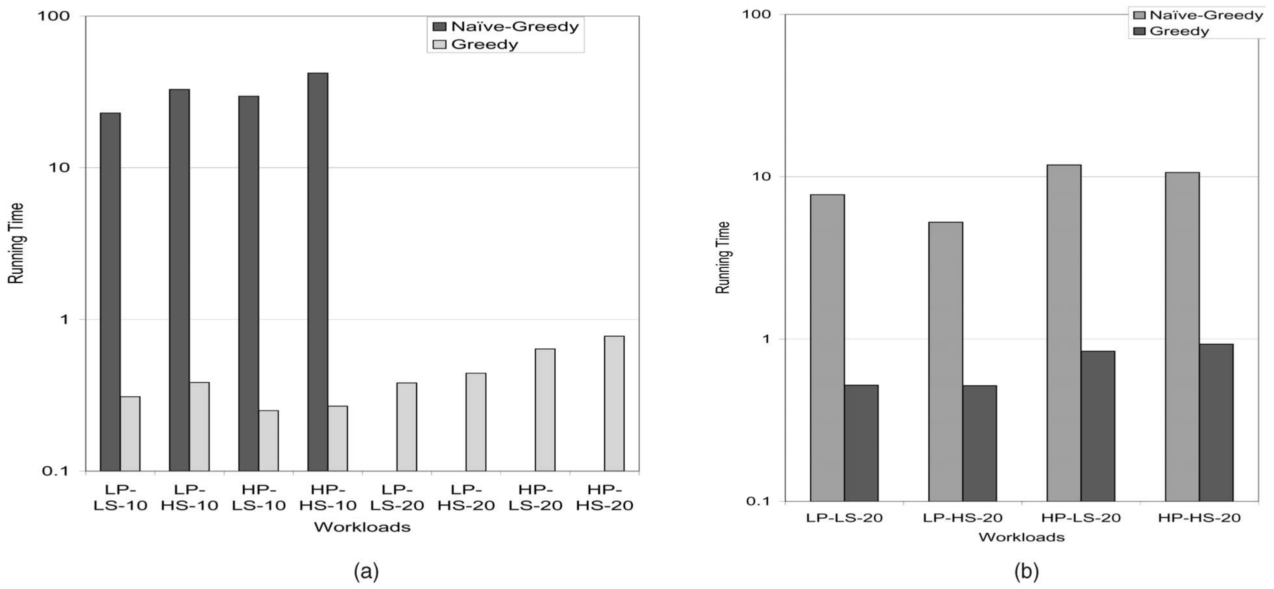


Fig. 5. Running time of Greedy and Naive Greedy on DBLP, normalized to Two-Step. (a) DBLP. (b) Movie.

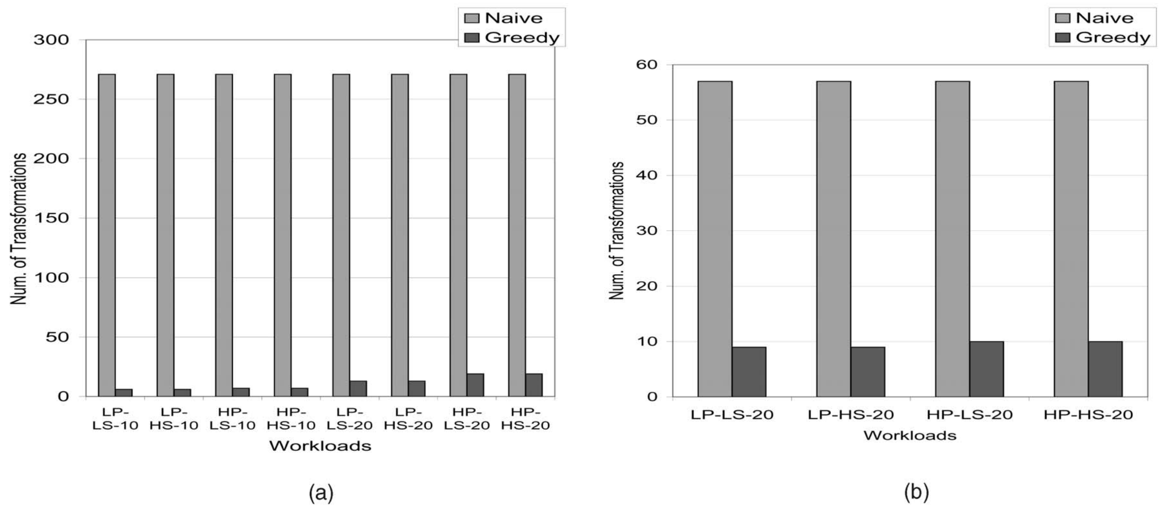


Fig. 6. Number of transformations searched by Greedy and Naive Greedy on DBLP. (a) DBLP. (b) Movie.

proposed in Section 4. The results demonstrate that ignoring subsumed transformations is the major factor of speed-up (ranges from 8 to 12). The other candidate selection rules bring about a further factor of 2 speed-up. The results show the importance of considering only the mappings not subsumed by physical design. Moreover, we observe no quality drop due to candidate selection for all workloads.

5.3.2 Effectiveness of Candidate Merging

Fig. 8 reports the query execution time and the running time of Greedy algorithm with three different merging strategies for 20-query workloads. The merging strategies are greedy merging, no merging, and exhaustive merging that enumerate all possible merged candidates. The results show the importance of candidate merging: The cost of results without merging is substantially higher than results with merging (the difference is on average a factor of 2). The results also show the greedy merge algorithm is effective because it has about the same quality as the exhaustive merging, but is 2 to 10 times faster, and is almost as fast as no merging. We find that greedy merging tends to merge candidates on relations with large sizes referred in expensive queries, and such transformations account for most performance improvement. We also find our heuristic cost model works reasonably well, especially when the query plans contain table scan or index scan operators because such operators' cost is often proportional to relation sizes.

5.3.3 Effectiveness of Cost Derivation

Fig. 9a reports the query execution time of the mappings returned by Greedy algorithm using or not using the cost derivation for the 20-query workloads. The running time is reported in Fig. 9b. The results show that cost-derivation leads to little drop of quality (up to 3 percent of the hybrid inlining mapping cost), but it speeds up the algorithm by a factor of 4 to 10. This confirms that the physical design tool and query optimizer demonstrate some stable property

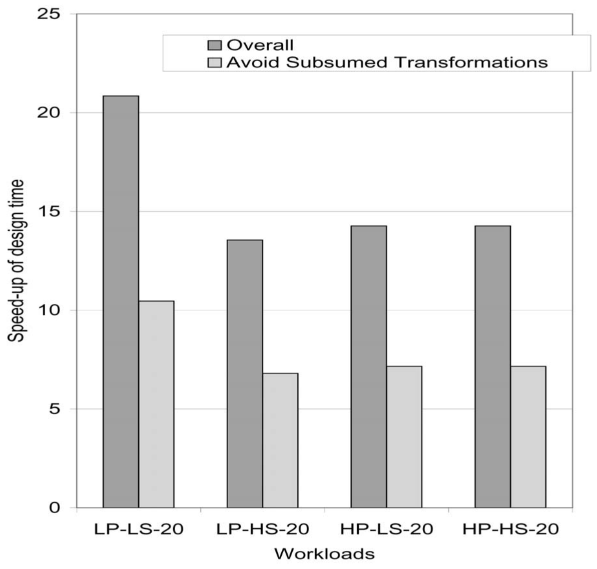


Fig. 7. Speed-up due to candidate selection on DBLP data.

such that the same indexes and materialized views are often recommended for the cases we use cost-derivation in Section 4, and the same query plans are selected by the optimizer.

6 Related Work

There has been lots of work on XML-to-relational storage mapping [5], [9], [9], [12], [15], [18], [19], [20], [22]. The NP hardness of the problem is proved in [16]. Deutsch et al. [10] proposed to use data mining techniques to identify frequently occurring portions of XML documents and store them in relations. A DTD-based approach was proposed in [20], and several heuristic mappings were proposed. An Edge-based approach was proposed for schema-less XML data in [12]. Bohannon et al. proposed a cost-based approach to find a relational schema using workload information [5]. Ramanath et al. further improved their algorithm by proposing a few new types of transformations (type split/merge, etc.). They also considered the interactions between logical design transformations and proposed the deep merge technique in [18]. However, all of the above work has not addressed the interplay of logical mapping and physical design. As described in Section 5, solutions ignoring such interplay can suffer in quality and efficiency.

Most major commercial relational database systems [4], [8] provide a default way to map XML documents to relational storage and allow users to specify their own mappings, which is often quite tedious given the complexity of the problem. Our work can be used to guide and advise the users.

Physical database design has been examined significantly [2], [7], [23]. However, our focus is on storing XML in relational databases, and we use existing physical design techniques. The optimization techniques (candidate selection, candidate merging, and cost derivation) we propose have the same spirit as those used in index and materialized view selection [2], [7]. However, our techniques are specific for XML-to-relational mapping and exploit the properties of mappings and their interplay with physical design.

Translating XML queries to SQL statements were studied in [11], [21]. Our focus is on combining logical and physical design, and we use existing query translation techniques [21]. In [16], [17], the authors found that there exists interaction of logical design with query translation, but physical design is not considered. Further understanding of the interaction of logical and physical design with query translation will be future work.

There also exists work on XML specific indexing structures [24] and query processing techniques [6]. However, our focus is to use relational techniques.

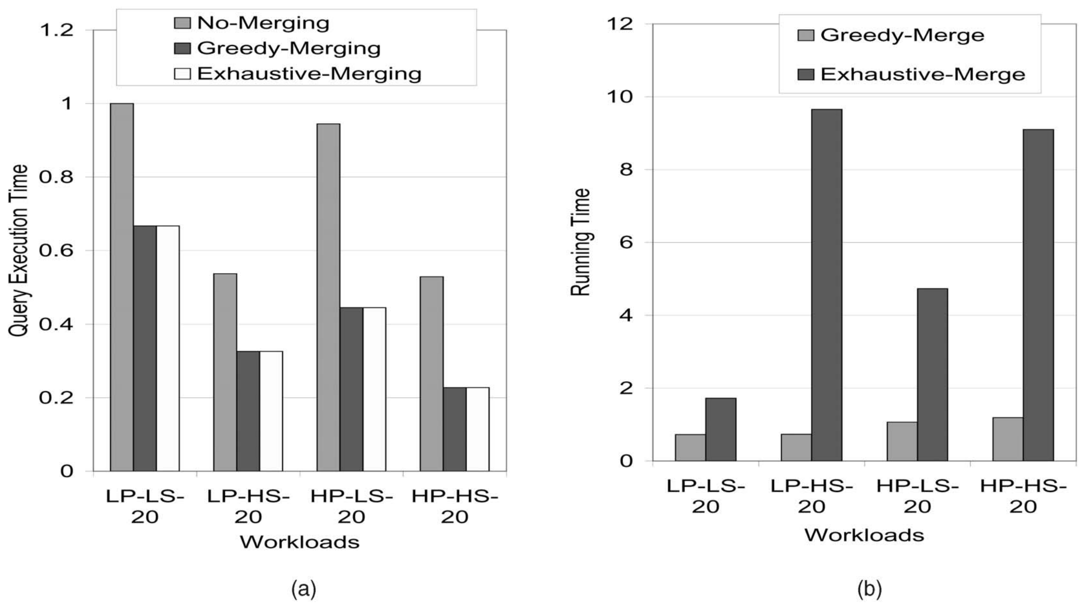
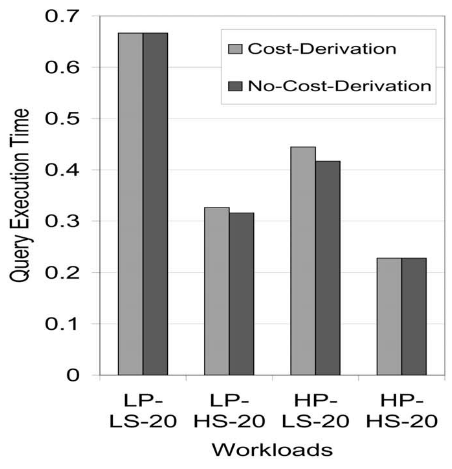
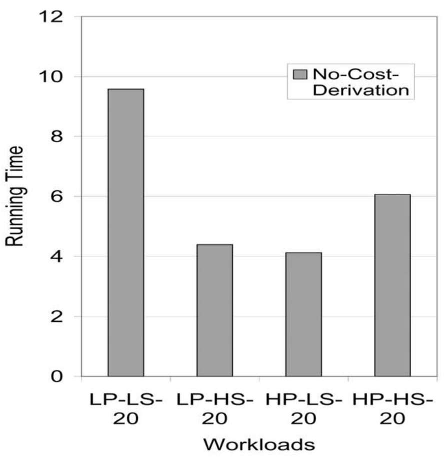


Fig. 8. (a) Query execution time (normalized to hybrid inlining mapping) and (b) algorithm running time (normalized to no merging) of Greedy on DBLP with various merging algorithms.



(a)



(b)

Fig. 9. (a) Query execution time (normalized to hybrid inlining mapping) and (b) algorithm running time (normalized to with cost derivation) of Greedy on DBLP with and without cost-derivation.

7 Conclusions

In this paper, we find that optimizing the "shredding" of XML data in relational databases must take the interplay of logical and physical design into account. We point out that existing logical design transformations can be divided into two categories, those are subsumed by physical design and those are not. Transformations in the first category may be combined with those in the second category to generate mappings not considered by physical design. We further proposed an efficient search algorithm that uses workload-based pruning and avoids searching mappings considered by physical design. Our experiments demonstrated that the proposed algorithm recommends significantly better combination of logical and physical design compared to some of the existing algorithms (Two-Step) and takes significantly less time to find the recommendation compared to the rest of existing algorithms (Naive-Greedy). For future work, we plan to consider more general XML queries (including update queries).

Acknowledgments

The work was done when Z. Chen worked at Microsoft Research.

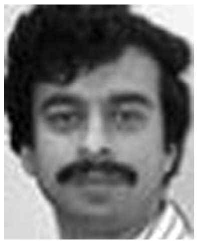
References

[1] DBLP, XML records, http://dblp.uni-trier.de/xml/, 2005.
[2] S. Agrawal, S. Chaudhuri, and V.R. Narasayya, "Automated Selection of Materialized Views and Indexes in SQL Databases," *Proc. Very Large Data Bases Conf.*, 2000.
[3] S. Agrawal, V.R. Narasayya, and B. Yang, "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design," *Proc. ACM SIGMOD*, pp. 359-370, 2004.
[4] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy, "Oracle8i—The XML Enabled Data Management System," *Proc. Int'l Conf. Data Eng.*, 2000.
[5] P. Bohannon, J. Freire, P. Roy, and J. Simeon, "From XML Schema to Relations: A Cost-Based Approach to XML Storage," *Proc. Int'l Conf. Data Eng.*, 2002.
[6] N. Bruno, N. Koudas, and D. Srivastava, "Holistic Twig Joins: Optimal XML Pattern Matching," *Proc. ACM SIGMOD*, 2002.
[7] S. Chaudhuri and V.R. Narasayya, "An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server," *Proc. Very Large Data Bases Conf.*, 1997.
[8] J.M. Cheng and J. Xu, "XML and DB2," *Proc. Int'l Conf. Data Eng.*, 1999.
[9] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From Structured Documents to Novel Query Facilities," *Proc. ACM SIGMOD Conf.*, 1994.
[10] A. Deutsch, M.F. Fernandez, and D. Suciu, "Storing Semistructured Data with STORED," *Proc. ACM SIGMOD Conf.*, 1999.
[11] M.F. Fernandez, A. Morishima, and D. Suciu, "Efficient Evaluation of XML Middle-Ware Queries," *Proc. ACM SIGMOD Conf.*, 2001.
[12] D. Florescu and D. Kossmann, "Storing and Querying XML Data Using an RDBMS," *IEEE Data Eng. Bull.*, 1999.
[13] J. Freire, J.R. Haritsa, M. Ramanath, P. Roy, and J. Simon, "StatiX: Making XML Count," *Proc. ACM SIGMOD Conf.*, pp. 181-191, 2002.
[14] H.V. Jagadish, S. Al-Khalifa, A. Chapman, L.V.S. Lakshmanan, A. Nierman, S. Paparizos, J.M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu, "Timber: A Native XML Database," *VLDB J.*, vol. 11, no. 4, 2002.
[15] M. Klettke and H. Meyer, "XML and Object-Relational Database Systems—Enhancing Structural Mappings Based on Statistics," *Proc. Third Int'l Workshop Web and Databases*, 2000.
[16] R. Krishnamurthy, V.T. Chakaravarthy, and J.F. Naughton, "Difficulty of Finding Optimal Relational Decompositions for XML Workloads: A Complexity Theoretic Perspective," *Proc. Int'l Conf. Database Theory*, 2003.
[17] R. Krishnamurthy, R. Kaushik, and J.F. Naughton, "Efficient XML-to-SQL Query Translation: Where to Add the Intelligence?" *Proc. Very Large Data Bases Conf.*, pp. 144-155, 2004.
[18] M. Ramanath, J. Freire, J.R. Haritsa, and P. Roy, "Searching for Efficient XML-to-Relational Mappings," *Xsym*, pp. 19-36, 2003.
[19] A. Schmidt, M.L. Kersten, M. Windhouwer, and F. Waas, "Efficient Relational Storage and Retrieval of XML Documents," *Proc. Third Int'l Workshop Web and Databases*, 2000.
[20] J. Shanmugasundaram, G. He, K. Tufte, C. Zhang, D. DeWitt, and J. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," *Proc. Very Large Data Bases Conf.*, 1999.
[21] J. Shanmugasundaram, E.J. Shekita, R. Barr, M.J. Carey, B.G. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," *Proc. Very Large Data Bases Conf.*, 2000.
[22] T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and Retrieval of XML Documents Using Object-Relational Databases," *Proc. 10th Int'l Conf. and Workshop Database and Expert Systems Applications*, 1999.

[23] G. Valentin, M. Zuliani, D.C. Zilio, G.M. Lohman, and A. Skelley, "DB2 Advisor: An Optimizer Smart Enough to Recommend Its Own Indexes," *Proc. Int'l Conf. Data Eng.*, 2000.

[24] H. Wang, S. Park, W. Fan, and P.S. Yu., "Vist: A Dynamic Index Method for Querying XML Data by Tree Structures," *Proc. ACM SIGMOD Conf.*, 2003.

[25] World Wide Web Consortium, XML Schema, 2001, http://www.w3.org/XML/Schema.



Surajit Chaudhuri received the PhD degree from Stanford University in 1991 and worked at Hewlett-Packard Laboratories, Palo Alto from 1991-1995. He leads the Data Management and Exploration Group at Microsoft Research http://research.microsoft.com/users/surajitc. In 1996, Surajit started the AutoAdmin project on self-tuning database systems at Microsoft Research and developed novel automated physical design tuning technology for SQL Server 7.0, SQL Server 2000, and SQL Server 2005. More recently, he has begun work in the area of data cleaning and integration techniques. Part of this research has been incorporated in Micrsooft SQL Server 2005. Dr. Chaudhuri is also interested in the problem of querying and discovery of information in a flexible manner exploiting text search as well as DBMS querying functionality. He was awarded the 2004 ACM SIGMOD Contributions award for developing the Conference management Service at Microsoft Research. He is a member of the IEEE.

Zhiyuan Chen received the PhD degree in computer science from Cornell University in 2002. Presently, he is an assistant professor in the Information Systems Department at the University of Maryland, Baltimore County. His research interests include XML and semistructured data, privacy-preserving data mining, data integration, automatic database tuning, and database compression. He is a member of ACM and IEEE.

Kyuseok Shim received the BS degree in electrical engineering from Seoul National University in 1986, and the MS and PhD degrees in computer science from the University of Maryland, College Park, in 1988 and 1993, respectively. He is currently an associate professor at Seoul National University, Korea. Previously, he was an assistant professor at the Korea Advanced Institute of Science and Technology (KAIST), Korea. Before joining KAIST, he was a member of technical staff (MTS) and one of the key contributors to the Serendip data mining project at Bell Laboratories. He also worked for Quest Data Mining project at IBM Almaden Research Center. Dr. Shim has been working in the area of databases focusing on data mining, bioinformatics, data warehousing, query processing and query optimization, XML, and semistructured data. He is currently on the editorial boards of the *VLDB Journal* and the *IEEE Transactions on Knoweldge and Data Engineering*. He has served as a program committee member for ACM SIGKDD, ACM SIGMOD, ICDE, ICDM, PAKDD, and VLDB conferences. He is a member of the IEEE.

Yuqing Wu received the BS and MS degrees from Peking University, China, and the PhD degree in computer science from the University of Michigan, Ann Arbor, in 2004. is an assistant professor in the School of Informatics and adjunct assistant professor in the Computer Science Department, Indiana University, Bloomington. Dr. Wu is one of the founders of the TIMBER project under development at the University of Michigan, a high-performance native XML database system capable of operating at large scale, through use of a carefully designed tree algebra and judicious use of novel access methods and optimizations techniques. Her research covers XML data storage, XML indexing, query processing, query optimization, query parsing and rewriting, and focuses on cost-based query optimization of XML queries. She is also involved in research related to data mining, Web data integration, and data extraction. She is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.