On the Difficulty of Finding Optimal Relational Decompositions for XML Workloads: a Complexity Theoretic Perspective

Rajasekar Krishnamurthy **, Venkatesan T. Chakaravarthy ***, and Jeffrey F. Naughton **

University of Wisconsin, Madison, WI 53706, USA,
{sekar,venkat,naughton}@cs.wisc.edu

Abstract. A key problem that arises in the context of storing XML documents in relational databases is that of finding an optimal relational decomposition for a given set of XML documents and a given set of XML queries over those documents. While there have been a number of ad hoc solutions proposed for this problem, to our knowledge this paper represents a first step toward formalizing the problem and studying its complexity. It turns out that to even define what one means by an optimal decomposition, one first needs to specify an algorithm to translate XML queries to relational queries, and a cost model to evaluate the quality of the resulting relational queries. By examining an interesting problem embedded in choosing a relational decomposition, we show that choices of different translation algorithms and cost models result in very different complexities for the resulting optimization problems. Our results suggest that, contrary to the trend in previous work, the eventual development of practical algorithms for finding relational decompositions for XML workloads will require judicious choices of cost models and translation algorithms, rather than an exclusive focus on the decomposition problem in isolation.

1 Introduction

In order to leverage existing investments in relational database technology, there has recently been considerable interest in storing XML documents in relational databases. This problem has two main parts to it: (i) Given an XML schema (and possibly a query workload and statistics), choose a good relational decomposition and (ii) Given the XML schema and the corresponding relational decomposition, translate XML queries to SQL over this relational schema. While the two problems have been studied independently [3, 4, 6, 10, 12], they are actually closely related. In this paper, we study the relationship between the two problems, namely, choosing a good relational decomposition and using a good query translation algorithm.

We show, through experiments with a commercial RDBMS and a well-known XML benchmark, that there exist translation algorithms T_1 and T_2, and relational decompositions D_1 and D_2, such that with translation T_1 decomposition

** Research supported in part by NSF grants CSA-9623632 and ITR-0086002
*** Research supported in part by NSF grants

D_1 is better than D_2, while with translation T_2 decomposition D_2 is better than D_1. This implies that one cannot talk about the quality of a decomposition without discussing the query translation algorithm to be used.

Any algorithm that attempts to find the optimal relational decomposition will have some cost model for the resulting relational queries, since it needs some way of evaluating the quality of the decomposition. To talk about the difficulty of finding good relational decompositions, we need to be specific about the cost models used. To show that the choice for the cost model can have a profound impact on the complexity of finding the optimal relational decomposition, we introduce two simple cost metrics and explore the complexity of the problem with each in turn.

Describing and analyzing the interaction between decompositions and query translations in full generality is a daunting task that appears (at least to us) unlikely to be amenable to a clean formal analysis; accordingly, here we consider a subset of the problem that is constrained enough to be tractable yet rich enough to provide insight into the general problem. We specify this subset by identifying a subset of XML schemas and a restricted class of XML queries over these schemas. We also specify a class of decompositions that, while not fully general, covers a wide range of decompositions we have seen in literature. We identify an important subproblem that must be addressed when designing an XML-to-Relational decomposition, which we call the *Grouping* problem. The subset of schemas and queries we consider captures the crux of the interaction between the decompositions and query translation algorithms in the context of the *Grouping* problem. We then present three query translation algorithms, *NaiveTranslation*, *SingleScan* and *MultipleScan*. In this setting, we look at how the complexity of choosing a good solution varies as we choose different combinations of translation algorithms and cost models.

Finally, we describe the *CompleteGrouping* problem, where we no longer fix the query translation algorithm, so the goal is to find the best pair of (relational decomposition strategy, query translation algorithm). We analyze the complexity of this problem for the two cost metrics.

The rest of the paper is organized as follows. We first show in Section 2 how the relative performance of different decompositions varies as the translation algorithm is varied. We also present the two cost metrics used in this paper. We then present a formal model describing the various parameters in the problem in Section 3 and formally define the *Grouping* problem. We describe the set-system coloring problem in Section 4 and show how it corresponds to the *Grouping* problem for the family of instances we consider. We also prove some complexity results for set-system coloring. We discuss the complexity of the *Grouping* problem in Section 5 as the query translation algorithm and cost model are varied and present our conclusions.

2 Relational Decompositions and Query Translation Algorithms

In this section, using the XMark benchmark XML schema [11], we illustrate the interaction between relational decompositions and XML-query to relational-

query translation algorithms. We then describe two cost metrics used to compare alternative decompositions.

2.1 Motivating Example

In this section, we show how the quality of a decomposition is dependent on the query translation algorithm used. A part of the XMark schema is presented in Figure 1. Consider the various *Item* elements and the corresponding *InCategory* elements that appear in the schema. The techniques proposed in existing literature [3, 12] map these elements into relations in one of two ways. The first way is to create six Item relations, one for each occurrence of Item in the schema. The Item elements are stored in relations AfricaItem, AsiaItem and so on based on their parent element. Corresponding *InCategory* relations are also created. Let us call this the *fully partitioned* strategy. The second way is to create an *Item* relation and an *InCategory* relation and store all the Item elements and their categories in these relations respectively. Let us call this the *fully grouped* strategy. Informally, how we decide to group the Item and InCategory elements into one or more relations is the *Grouping* problem. Consider the path expression query Q in Figure 2. Let us now look at how this query is translated into a relational query. Consider the following simple algorithm. Identify all paths in the schema that satisfy the query. For each path, generate a relational query by joining all relations appearing in this path. The final query is the union of the queries over all satisfying paths (six paths for Q). This query translation algorithm is presented in Section 3.2. The relational queries for the fully-partitioned and fully-grouped strategies, XQ^1_{fp} and XQ^1_{fg} respectively, are given below. Notice how the two queries are very similar and differ mainly in the relations involved.

XQ^1_{fp}:

```
select count(*)
from   Site S, AfricaItem I1,
       AfricaInCategory C1
where  S.id = I1.parent
       and I1.id = C1.parent
       and C1.category = 'cat1'
union all ... (6 queries)
```

XQ^1_{fg}:

```
select count(*)
from   Site S, Item I, InCategory C
where  S.id = I.parent
       and I.id = C.parent
       and I.region = 'africa'
       and C.category = 'cat1'
union all ... (6 queries)
```

We know that, using the fully grouped strategy, relation InCategory contains exactly the set of elements returned by the path expression "/Site/Regions//Item/InCategory". So, we can translate query Q into a selection query on relation InCategory. In a similar fashion, we can simplify the relational query for the fully partitioned strategy as well. This kind of optimizations can be performed by using the XML schema and XML-to-Relational mapping information. Two such query translation algorithms are presented in Section 3.2 and 3.2. The resultant relational queries in this case, XQ^2_{fp} and XQ^2_{fg}, are given below.

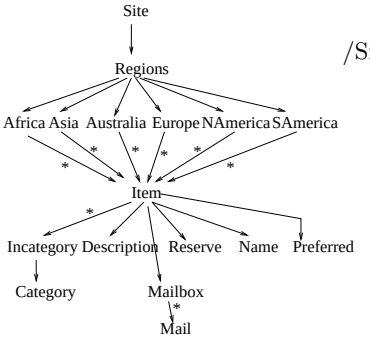

Fig. 1. XMark Benchmark schema

Find the number of items in a given category:
/Site/Regions//Item/InCategory[@Category = 'cat1']

Fig. 2. Path expression query Q

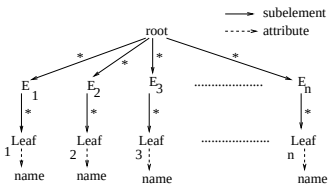

Fig. 3. Sample XML Schema in $\mathcal{T}$

XQ^2_{fp}**:**

```
select count(*)
from   AfricaInCategory C1
where  C1.category = 'cat1'
union all ... (6 paths)
```

XQ^2_{fg}**:**

```
select count(*)
from   InCategory C
where  C.category = 'cat1'
```

On the 100MB XMark dataset [11], we noticed that XQ^1_{fg} was 50% slower than XQ^1_{fp}, while XQ^2_{fg} was about three times faster than XQ^2_{fp}. So, we see that for query Q, with algorithm *NaiveTranslation*, the *fully partitioned* strategy is better, whereas with algorithm *MultipleScan*, the *fully grouped* strategy is better. As a result, the quality of a decomposition is closely related to the query translation algorithm used.

2.2 Two Simple Cost Metrics

An algorithm that attempts to find the optimal relational decomposition needs a cost model for comparing alternative decompositions. One of our goals in this paper is to show that the choice of the cost metric has a big impact on the complexity of the problem. To illustrate this fact, we look at two simple cost models for relational queries.

- ***RelCount*:** The cost of a relational query is the number of relation instances in the relational algebra expression. This metric is inspired by historical work on minimizing unions of conjunctive queries [9].
- ***RelSize*:** The cost of a relational query is the sum of the number of tuples in relation instances in the relational algebra expression.

We next present some example scenarios where the above metrics are applicable. Consider the relational queries generated by algorithm *MultipleScan* for query Q.

The two queries, XQ^2_{fg} and XQ^2_{fp}, have a selection predicate on the category attribute. If this predicate is a highly selective predicate and clustered indices are

present on the category columns of these relations, then the relational optimizer will choose an index lookup plan for these queries. For query XQ^2_{fg}, the cost will be the sum of the cost of an index lookup and the cost for fetching all satisfying tuples. For query XQ^2_{fp}, it will be the sum of the cost of the six subqueries, each of which is the sum of the cost of an index lookup and the cost for fetching all satisfying tuples. Since, the query is highly selective, we can assume that the index lookup cost is the dominating cost and so XQ^2_{fp} is six times more expensive than XQ^2_{fg}. In this scenario, *RelCount* is a good cost metric.

On the other hand, if the selection predicate is not very selective, then the optimizer may choose a plan that scans the relations. In this case, the cost of XQ^2_{fg} depends on the size of the InCategory relation, while the cost of XQ^2_{fp} depends on the sum of the sizes of the six region InCategory relations. In this scenario, *RelSize* is a good cost metric.

We would like to emphasize the fact that the purpose of these cost models is not to claim that they are optimal, or even very good in most cases, but rather that they are at least reasonable and they can be used to show how the complexity of the problem depends upon the chosen cost model.

3 Formal Model

To study the complexity of finding the optimal relational decomposition and how it depends on the query translation algorithm used and the cost model, we next formalize the problem in this section. We limit ourselves to a subset of the full problem by fixing the class of XML schemas and path expression queries that we consider. We then describe the class of relational decompositions, the class of relational queries that can be produced by the query translation algorithms and three candidate translation algorithms for the class of path expression queries we consider. Finally we define the *Grouping* problem, a subproblem lurking in the decomposition problem, which will be used to explore the complexity of the decomposition problem.

3.1 Definitions

XML Schema: We consider the schema $\mathcal{T}$, given in Figure 3. By varying n, the number of children of the root, we get a family of instances. Let *Label(v)* denote the name of the element v. If an edge is labeled with a "*", then the child element may occur more than once per parent element in the data. Otherwise the child element occurs exactly once per parent element. We refer to elements in the schema as *elements* and the corresponding elements in XML documents as *instance elements*. We refer to the element in $\mathcal{T}$ that corresponds to $/root/E_i/Leaf$ as element $Leaf_i$ or simply as i. Let Parent($Leaf_i$) denote the element E_i, which is the parent of $Leaf_i$.

Path expression queries: We consider a class of path expression queries, $\mathcal{Q}$, that select a subset of the *Leaf* elements. A query $Q \in \mathcal{Q}$ is of the form $/\text{root}/(e_1|e_2|\ldots|e_k)/Leaf[@name = value]$, where e_i is some E_j. The query Q selects a subset of the *Leaf* elements from the schema based on the path conditions in Q. We call this subset of *Leaf* elements *Range(Q)*. For example, for

Q = /root/$(E_2|E_5|E_7)$/*Leaf*[@*name* = *value*], Range(Q) = $\{2, 5, 7\}$. Applying the value conditions on the instance elements corresponding to elements in Range(Q) will evaluate the result for Q.

Relational Decompositions: A number of relational decompositions have been proposed for XML data in literature [3, 4, 6, 10, 12]. These techniques can be broadly classified into two categories: (i) methods that use the XML schema information to decide on the relational schema [3, 12] and (ii) methods that decide a relational schema independent of the XML schema [4, 6, 10]. For ease of exposition, we consider only the former techniques in this paper. We have extended Theorems 9-12 to the latter techniques as well.

As an example of decompositions based on the XML schema, for the schema in Figure 3, the *Shared* approach [12] will create $n + 2$ relations: one for storing *root* elements, one for each of the n E_i elements and one for storing all the *Leaf* elements (say relation L). The name attribute is stored in relation L along with its parent *Leaf* element. The relation L will have columns (*Leaf, name, parentid*) corresponding to the *Leaf* element, name attribute and information about the parent E_i element.

In general, a relational decomposition for the XML schema can be viewed as a mapping function, σ, from vertices in $\mathcal{T}$ to a tuple (R, C). The notation $\sigma(x) = (R_1, C_1)$ implies that instance elements of element x are stored in column C_1 of relation R_1. Since σ is a function, all instance elements of a single element are all stored in the same relation. Moreover, two elements in the XML schema can be mapped to the same column of a relation only if they have the same label. For example, $Leaf_1$ and $Leaf_2$ can be mapped to the same relational column, but $Leaf_1$ and the corresponding *name* attribute cannot be mapped to the same relational column.

Relational queries : We consider the translation of the queries in $\mathcal{Q}$ to equivalent relational queries containing the select, project, join and union operators from relational algebra. These relational queries can also be viewed as the union of several conjunctive queries. As a technical detail, we do not allow disjunctions in selection and join conditions because our simple cost metrics do not capture the actual costs of relational queries in this scenario. This restriction can be lifted, but at the expense of requiring more expensive cost metrics and our results still hold. In this paper, for clarity of exposition, we restrict the class of operators allowed and use simple cost metrics. We also do not allow the set difference operator in the relational queries; the impact of lifting this restriction is an interesting topic for future work.

A Query Q in $\mathcal{Q}$ selects one or more of the *Leaf* elements in $\mathcal{T}$. So, the equivalent relational query will be the union of one or more queries Q_i, where each Q_i is a selection query or has a single join condition. If two *Leaf* elements $Leaf_i$ and $Leaf_j$ are mapped to the same relation R and Range(Q) contains just one of them (say $Leaf_i$), the relational query joins the relation R with $\sigma(E_i)$ (E_i is the parent of $Leaf_i$). For example, under the *fully grouped* decomposition represented by mapping σ, the query $/root/E_1/Leaf$ will translate into the query $\sigma(E_1) \bowtie \sigma(Leaf_i)$.

The cost of a path expression query Q for a given query translation algorithm is the cost of the relational query generated by the algorithm for Q.

The following definition will be useful in discussing later results.

DEFINITION **1** *A set S of elements in $\mathcal{T}$ is called a* scannable *set under a mapping σ if all the elements in S have the same label and are mapped to the same relational column (R, c) and no other element $v \notin S$ is mapped to (R, c).*

Intuitively if S is a *scannable* set under σ, then a query Q having $\text{Range}(Q) = S$ can be translated into a relational selection query on R.

The queries in $\mathcal{Q}$ have a selection condition on the *name* attribute of *Leaf* elements. So, it can be shown that storing the *Leaf* element and its *name* attribute in the same relation is better than storing them in two different relations. Similarly, it can be shown that decompositions that map two *Leaf* elements to two different columns of the same relation, have an equally good strategy that maps the *Leaf* elements to two different relations. This is due to the fact that disjunctions are not allowed in the relational queries. So, without loss of generality, we can assume that two *Leaf* elements are mapped to the same relation only if they are mapped to the same column[1].

We can also show that for the given family of instances $(\mathcal{T}, \mathcal{Q})$, under our cost metric, different mappings for the E_i and *root* elements have the same performance. What really makes the difference is how the n *Leaf* elements are stored in k relations, for some $1 \leq k \leq n$.

The space of relational decompositions that needs to be examined to find the optimal decomposition consists of the various ways in which the *Leaf* elements can be grouped into k relations. We next formally define the *Grouping* problem and the *CompleteGrouping* problem.

DEFINITION **2** *Given a schema $\mathcal{T}$, a query workload $Q_W \subseteq \mathcal{Q}$ and an XML-to-relational query translation procedure, the* Grouping *problem is to find a partitioning of the n Leaf elements into k relations ($1 \leq k \leq n$) such that the sum of the cost of all queries in Q_W is minimized.*

DEFINITION **3** *Given a schema $\mathcal{T}$ and a query workload $Q_W \subseteq \mathcal{Q}$, the* CompleteGrouping *problem is to find the partitioning of the n Leaf elements into k relations ($1 \leq k \leq n$) and the XML-to-relational query translation such that the sum of the cost of all queries in Q_W is minimized.*

3.2 Query Translation Algorithms

In this section, we briefly look at three different algorithms for translating a path expression query Q into SQL. The *NaiveTranslation* algorithm was first presented in [12]. The other two algorithms are improvements over this algorithm for the class of path expression queries.

Let $\text{Range}(Q) = S = \{v_1, v_2, \ldots, v_m\}$.

[1] Notice that this automatically rules out solutions like a Universal relation for all the n *Leaf* elements.

Procedure NAIVETRANSLATION(S)

1 Let $\sigma(root) = (R_1, C_1)$.
  Set Result = *null*
2 For $i = 1$ to m
3 Let $\sigma(Parent(v_i)) = (R_2, C_2)$,
  $\sigma(v_i) = (R_3, C_3)$.
4 Result = Result $\cup$
  $\Pi_{C_3}(R1 \bowtie R_2 \bowtie R_3)$.

Fig. 4. ***NaiveTranslation*** **algorithm**

Procedure *SingleScan*(S)

1 Set Result = *null*
2 If (S, σ) is a *scannable* set, let (R_1, C_1)
  be the corresponding relational column
3 Result = $\Pi_{C_1}(R_1)$.
4 Otherwise,
5 For $i = 1$ to m
6 Let $\sigma(Parent(v_i)) = (R_2, C_2)$,
  and $\sigma(v_i) = (R_3, C_3)$.
7 Result = Result $\cup$ $\Pi_{C_3}(R_2 \bowtie R_3)$.

Fig. 5. ***SingleScan*** **algorithm**

Procedure *MultipleScan*(S)

1 Partition S into equivalence classes $S_1, S_2, \ldots S_r$ based on the mapping σ.
2 Set Result = *null*
3 For $i = 1$ to r,
4 Let $S_i = \{v_1^i, v_2^i, \ldots, v_s^i\}$ and $\sigma(v_j^i) = (R_1, C_1)$.
5 If (S_i, σ) is a *scannable* set,
6 Result = Result $\cup$ $\Pi_{C_1}(R_1)$.
7 Otherwise,
8 For $j = 1$ to s
9 Let $\sigma(Parent(v_j^i)) = (R_2, C_2)$, .
10 Result = Result $\cup$ $\Pi_{C_1}(R_1 \bowtie R_2)$.

Fig. 6. ***MultipleScan*** **algorithm**

NaiveTranslation **Algorithm:** The naive translation strategy performs a join between the relations corresponding to all the elements appearing in a query. For example, let us consider the fully grouped strategy, where $\sigma(Leaf_i) = Leaf$. The query $/root/E_2/Leaf$ will be translated into a join among the *root*, E_2 and *Leaf* relations. A wild-card query will be converted into union of many queries, one for each satisfying wild-card substitution.

As an example, $//Leaf$ will be translated into the union of n queries, one for each *Leaf* element. Each of these queries will be a join between the *root*, E_i and $\sigma(Leaf_i)$ relations. Comparing this with the best translation scheme, which just performs a selection on the *Leaf* relation, we see that this scheme performs a lot of unnecessary computation. The translation procedure is presented in Figure 4. The condition on the attribute value is translated into a selection condition. This selection condition and the join conditions have been omitted in the above translation for clarity.

SingleScan **Algorithm:** This translation procedure converts Q into a selection query on a single relation, if Range(Q) forms a *scannable* set. Otherwise, a separate relational query Q_i is issued for each of the *Leaf* elements $Leaf_i$. Q_i joins all relations on the schema path until the least common ancestor of all the *Leaf* relations has been reached. In this case, there is a join between the

appropriate E_i and $Leaf_i$ relations. For example, for the fully grouped strategy, $//Leaf$ will be a scan on the *Leaf* relation. But, $/root/(E_1|E_2|E_3)/Leaf$ will be translated as the union of three queries, one each for the three paths. The translation algorithm is given in Figure 5.

***MultipleScan* Algorithm:** Consider a query Q with $\text{Range}(Q) = \{1, 2, 3, 4, 5\}$. Under a decomposition that groups *Leaf* elements $\{1, 2, 3\}$ and $\{4, 5, 6, \ldots, n\}$ into two different relations, the translation into a single selection will fail. So, under *SingleScan*, the relational query will be the union of 5 join queries. In *MultipleScan*, $\text{Range}(Q)$ is partitioned into equivalence classes, $\{1, 2, 3\}$ and $\{4, 5\}$, based on σ. Then, *SingleScan* is applied on each of these partitions and the resulting query is the union of these queries. In this case, $\{1, 2, 3\}$ gets translated into a single query, while $\{4, 5\}$ becomes the union of two queries and the translation of Q is the union of these three queries. The translation algorithm is given in Figure 6.

4 Set System Coloring

In this section we define an abstract problem that is useful in investigating the complexity of the grouping problem. The problem of assigning relations to the n *Leaf* elements can be viewed as assigning colors to the elements of a set of n elements. We call this problem the *set system coloring* problem. We define four different cost metrics, $SSCost_{RC}$, $MSCost_{RC}$, $SSCost_{RS}$ and $MSCost_{RS}$ and show that this problem is NP-Hard and MAXSNP-Hard under the first three metrics. The problem is trivially solved under metric $MSCost_{RS}$. We also give a 2-approximation algorithm under metric $SSCost_{RC}$.

DEFINITION **4** *A set-system $\mathcal{S}$ is 4-tuple $(A, \mathcal{H}, wt, mult)$, where $A = \{a_1, a_2, \ldots, a_n\}$ is a set, $\mathcal{H} \subseteq 2^A$ and $wt : A \longrightarrow \mathcal{Z}^+$ and $mult : \mathcal{H} \longrightarrow \mathcal{Z}^+$ are functions.*

A set-system can also be viewed as a hypergraph with weights on its vertices and hyperedges. So we call elements of A *vertices* and those of $\mathcal{H}$ *hyperedges*. The function wt assigns a positive integer weight to each vertex and *mult* represents the multiplicity of the hyperedges. For a set $X \subseteq A$ let $wt(X) = \sum_{a \in X} wt(a)$.

The insight behind the above definition is the natural correspondence between the set-system problem and the *Grouping* problem defined in Section 3.1. Note that the vertices here correspond to the n *Leaf* elements in the grouping problem and the hyperedges correspond to the queries in the workload. For a query Q, given a mapping σ, if $Leaf_i \in \text{Range}(Q)$ and $Leaf_i$ is not in a *scannable* set, then *MultipleScan* algorithm will output a join query for $leaf_i$. For metric *RelCount*, the number of joins performed in this query is captured by the weight on the vertex corresponding to $leaf_i$. Observe that the weight of each vertex is one for instances $(\mathcal{T}, \mathcal{Q})$. For metric *RelSize*, the size of the relations in the join query is captured by the wt function. Here we assume that the input also includes statistics about the cardinality of the elements corresponding to each schema node. *mult* represents the frequency of each query. Note that the

query workload was defined as a set in Section 3.1 and the definition can easily be extended to include a frequency for each query in the workload.

DEFINITION **5** *A coloring σ is a mapping $\sigma : A \longrightarrow [1..n]$, where $n = |A|$. With respect to σ, a subset $X \subseteq A$ is said to be* scannable *if all the elements in X have the same color and no element in $(A - X)$ has that color.*

The set-system coloring problem consists of finding a coloring of minimum cost. We can define a variety of optimization problems by supplying different cost metrics for computing the quality of a coloring for a set-system. For our results, four cost metrics are useful. We first define metrics $SSCost_{RC}$ and $SSCost_{RS}$. Then the other two metrics, $MSCost_{RC}$ and $MSCost_{RS}$, are defined in terms of these two metrics.

$SSCost_{RC}(\mathcal{S}, \sigma)$: Let G be the set of *scannable* sets in $\mathcal{H}$, with respect to σ. Each set in G contributes a cost of 1. For every set $h \in (\mathcal{H} - G)$, each element a in h contributes a cost of $wt(a) + 1$. Accounting for the multiplicities, we get $SSCost_{RC}(\mathcal{S}, \sigma) = \sum_{h \in G} mult(h) + \sum_{h \in (\mathcal{H}-G)} \sum_{a \in h} (wt(a) + 1) * mult(h)$.

$SSCost_{RS}(\mathcal{S}, \sigma)$: Let G be the set of *scannable* sets in $\mathcal{H}$, with respect to σ. Each set in G contributes a cost equal to its size. For every set $h \in (\mathcal{H} - G)$, each element a in h contributes a cost of $wt(a) + |\sigma(a)|$. Accounting for the multiplicities, we get
$SSCost_{RS}(\mathcal{S}, \sigma) = \sum_{h \in G} mult(h) * |h| + \sum_{h \in (\mathcal{H}-G)} \sum_{a \in h} (wt(a) + |\sigma(a)|) * mult(h)$.

$MSCost_{RC}(\mathcal{S}, \sigma)$ and $MSCost_{RC}(\mathcal{S}, \sigma)$: Partition each $h \in \mathcal{H}$ into n equivalence classes $E(h) = \{h_1, h_2, \ldots, h_n\}$, where $h_i = \{a | a \in h \text{ and } \sigma(a) = i\}$. Let $\mathcal{S}_1$ denote the set system obtained by replacing each $h \in \mathcal{H}$ by the set $E(h)$. Then $MSCost_{RC}(\mathcal{S}, \sigma) = SSCost_{RC}(\mathcal{S}_1, \sigma)$ and $MSCost_{RS}(\mathcal{S}, \sigma) = SSCost_{RS}(\mathcal{S}_1, \sigma)$.

The above cost metrics give instances of the set-system coloring problem that correspond to different instances of the relational decomposition problem. For example, $SSCost_{RC}$ gives an instance of set-system coloring problem that corresponds to the problem of finding a relational decomposition for the schema in Figure 3 under query translation algorithm *SingleScan* and cost metric *RelCount*. Similarly, $MSCost_{RC}$, $SSCost_{RS}$ and $MSCost_{RS}$ correspond to the pairs (*MultipleScan*,*RelCount*), (*SingleScan*,*RelSize*) and (*MultipleScan*,*RelSize*) respectively.

We present the following results about the complexity of this problem under each of the above cost metrics.

4.1 Complexity results under metric $SSCost_{RC}$

In this section, we look at the complexity of set-system coloring under cost metric $SSCost_{RC}$.

THEOREM **1** *Under metric $SSCost_{RC}$, set-system coloring is NP-Hard, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be w, for any constant $w \geq 0$.*

Proof. The problem of finding the vertex cover on 3-regular graphs is known to be NP-Complete [7]. We give a reduction from this problem.

Let $G = (V, E)$ be the input graph with n vertices and m edges. We output the set-system $\mathcal{S} = (A, \mathcal{H}, wt, mult)$. Let $A = E$ and $wt(a) = w$, for all a. For each $v \in V$, we add the set of (three) edges incident on v to $\mathcal{H}$. The multiplicity of any hyperedge is one. Notice that $|\mathcal{H}| = n$. We can show that G has a vertex cover of size $\leq k$ if and only if $\mathcal{S}$ has a coloring with cost $\leq n + (3w + 2) * k$.

Given a vertex cover VC of G of size k, we can construct a coloring σ that uses $(n-k+1)$ colors. For each vertex $v \notin VC$, we color the three edges incident on v by a unique color. For all the edges that are not yet colored, we use the $(n - k + 1)^{th}$ color. This coloring will have at least $(n - k)$ good hyperedges. Similarly, given a coloring with $(n - k)$ good hyperedges, the set obtained by adding the vertices corresponding to these hyperedges forms an independent set (of size $n - k$). Its complement is a vertex cover of size k. We have shown that the graph has a vertex cover of size k if and only if the set system has a coloring with at least $(n - k)$ good sets. Moreover, if a coloring has $(n - k)$ good sets, then its cost is $n + (3w + 2) * k$. This completes the proof.

As set-system coloring is NP-Hard, we explore the possibility of efficient approximation algorithms. We show that it can be approximated within a factor of 2. We also show that the problem is MAXSNP-Complete. Thus it cannot have a polynomial time approximation scheme (PTAS) unless NP=P.

THEOREM **2** *For metric $SSCost_{RC}$, set-system coloring can be approximated within a factor of 2.*

Proof. We use the 2-approximation algorithm [2] for the weighted vertex cover problem (WVC). In WVC, given a graph with weights associated each vertex, the problem is to find the vertex cover of minimum cost. The cost of a vertex cover is the sum of the weights of the vertices in it.

Let the input set-system be $\mathcal{S} = (A, \mathcal{H}, wt, mult)$. First construct a weighted graph $G = (V, E)$ as follows. For each hyperedge $h \in \mathcal{H}$, add two nodes v_h and v'_h to the graph G and add an edge between them. We call the node v'_h as the companion of v_h. Add an edge between two nodes v_s and v_t, if $s \cap t \neq \phi$. For each node v_h set its weight to $(wt(h) + |h|) * mult(h)$. For the companion nodes, set the weight to be 1.

We next apply the known 2-approximation algorithm for WVC on G. Let the vertex cover returned by it be VC. We can construct a coloring σ for $\mathcal{S}$ in the following manner. For each pair, v_h, v'_h at least one of them has to be in VC. If both are present, remove v'_h from VC. Now, for each node v'_h in VC, give a unique color to all the elements in h. Assign a common new color to all the elements in A that are not yet colored.

It can be shown that

$$cost(\sigma) \leq cost(VC) \leq 2 * cost(VC^*) \leq 2 * cost(\sigma^*)$$

where σ^* is an optimal coloring of $\mathcal{S}$ and VC^* is an optimal vertex cover of G.

A problem is in MAXSNP if and only if it has some constant factor approximation algorithm [8]. So, by Theorem 2, the set-system coloring is in MAXSNP.

The vertex cover problem on 3-regular graphs is known to be MAXSNP-Complete [1]. We can show that the reduction given in proof of Theorem 1 is actually an L-reduction. Thus we have the following theorem.

THEOREM **3** *Under metric $SSCost_{RC}$, set-system coloring is MAXSNP-Complete, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be w, for any constant $w \geq 0$.*

THEOREM **4** *Under metric $SSCost_{RC}$, set-system coloring cannot be approximated within a factor of 1.36 in polynomial time, unless NP=P.*

Proof. We can show that, for any constant $c > 1$, if there is a polynomial time algorithm that approximates the above coloring problem within a factor of c, then for any $\epsilon > 0$, we can approximate vertex cover within a factor of $c - \epsilon$. Using the inapproximability bound of 1.36 for vertex cover [5], we get the above result.

4.2 Complexity results for other cost metrics

We have the following results for the other three cost metrics. We omit the proofs for lack of space.

THEOREM **5** *Under metric $SSCost_{RS}$, set-system coloring is NP-Hard and MAXSNP-Complete, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be w, for any constant $w \geq 3$.*

THEOREM **6** *Under metric $SSCost_{RS}$, when the weights of vertices are $<= 1$ and the multiplicities of the hyperedges are restricted to be one, set-system coloring is in P*

THEOREM **7** *Under metric $MSCost_{RC}$, set-system coloring is NP-Hard and MAXSNP-Hard, even if the multiplicities of the hyperedges are restricted to be one and the weights of vertices are restricted to be w, for any constant $w \geq 0$.*

THEOREM **8** *Under metric $MSCost_{RS}$, the coloring that assigns a different color for each vertex is an optimal solution.*

5 Complexity of the *Grouping* and *CompleteGrouping* Problems

Let us now look at the complexity of the *Grouping* problem for different query translation schemes under the two cost metrics.

THEOREM **9** *When* NaiveTranslation *is the query translation algorithm,*

- *under metric* RelCount*, every relational decomposition is an optimal solution for the* Grouping *problem.*

– *under metric* RelSize*, the* fully partitioned *strategy is the optimal solution for the* Grouping *problem.*

Proof Sketch: Under the *NaiveTranslation* algorithm, for all decomposition strategies, the resultant relational query Q is the union of the same number of queries. Each of these queries are similar in terms of the number of relations in it and they differ only in the actual relations. For metric *RelCount*, since the cost depends only on the number of relations, the cost is identical for all strategies. The relations involved in the query for the *fully partitioned* strategy are no larger than relations in any other strategy. So, for metric *RelSize*, the *fully partitioned* strategy is optimal. □

In Section 4 we showed that the set system coloring problem is NP-Hard and MAXSNP-Hard, under the two cost metrics, $SSCost_{RC}$ and $MSCost_{RC}$. We saw how the set-system coloring problem corresponds to the *Grouping* problem for the schema $\mathcal{T}$ and query workload $\mathcal{Q}$, when the elements to be grouped are represented as the set elements and the queries are represented as hyperedges. The weights of the vertices are set to one and multiplicity of hyperedges are set to the relative frequency of each query. It can be seen that, for the cost metric *RelCount*, the two metrics $SSCost_{RC}$ and $MSCost_{RC}$ for set-system coloring correspond to costs under the translation schemes *SingleScan* and *MultipleScan*. Hence, we have the following theorem.

THEOREM **10** *Under metric* RelCount*, for translation algorithms* SingleScan *and* MultipleScan *finding the optimal solution for the* Grouping *problem is NP-Hard. The problem of grouping the nodes is MAXSNP-Hard.*

COROLLARY **1** *Under metric* RelCount*, the* Grouping *problem is NP-Hard and MAXSNP-Hard.*

The approximation algorithm for $SSCost_{RC}$ in Theorem 2 gives a 2-approximation algorithm for the *Grouping* problem under *SingleScan*. Even though we restricted our discussion in this paper to the family of schema $\mathcal{T}$, the grouping problem on a general XML schema can be represented as a set-system by setting the weights on the vertices of the set-system appropriately. The results for the set-system coloring will extend to these instances as well.

In a similar fashion, for metric *RelSize*, the results for set system coloring under metrics $SSCost_{RS}$ and $MSCost_{RS}$ carry over for the two translation algorithms and we have the following results.

THEOREM **11** *Under metric* RelSize*, for translation algorithm* SingleScan *finding the optimal solution for the* Grouping *problem is NP-Hard. The problem of grouping the nodes is MAXSNP-Hard.*

THEOREM **12** *Under metric* RelSize*, when* MultipleScan *is the query translation algorithm, the* fully partitioned *strategy is the optimal solution for the* Grouping *problem.*

Table 1. Complexity of *Grouping* problem

	NaiveTranslation	*SingleScan*	*MultipleScan*
RelCount	any solution optimal	MaxSNP-Complete	MaxSNP-Hard
RelSize	fully partitioned	MaxSNP-Hard	fully partitioned

It can be shown that *MultipleScan* is the optimal translation algorithm for the instances $(\mathcal{T}, \mathcal{Q})$. Hence, we have the following results.

LEMMA **1** *For the class of path expression queries $\mathcal{Q}$ for the schema $\mathcal{T}$,* MultipleScan *produces the relational query with minimum cost.*

THEOREM **13** *Under metric* RelCount*, the* CompleteGrouping *problem is NP-Hard and MAXSNP-Hard.*

THEOREM **14** *Under metric* RelSize*, the pair (fully partitioned strategy,*MultipleScan*) is an optimal solution to the* CompleteGrouping *problem.*

In summary, we see that the complexity of the *Grouping* problem depends a lot on the query translation algorithm used and the cost model that is appropriate for the given workload.

6 Conclusions and Future Work

In this paper, we considered the problem of finding optimal relational decompositions for XML workloads in a formal perspective. We identified the *Grouping* problem, an important sub-problem in choosing a good relational decomposition, and analyzed the complexity of this problem for three different query translation algorithms and two simple cost metrics. The results are summarized in Table 1. These results show that the query translation algorithm and the cost model play a vital role in the choice of a good decomposition, and choices for these dramatically affect the complexity of the problem.

Our work in this paper represents a first step toward formalizing and analyzing the problem of mapping XML data and queries to relational counterparts. Substantial room for future work exists in almost all directions. For example, it would be interesting to study the complexity of the problem for more elaborate cost functions, or relational queries that involve the difference operator, or more general classes of XML queries and schemas. It is our hope that the insight derived from our work here will be useful in such a study. Finally, we hope that the work presented here can serve as the basis for further work eventually leading to practical algorithms for choosing good relational decompositions for XML workloads.

References

1. P. Alimonti and V. Kann. Hardness of approximating problems on cubic graphs. In *Proc. 3rd Italian Conf. on Algorithms and Complexity, Lecture Notes in Computer Science, 1203*, pages 288–298. Springer-Verlag, 1997.
2. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–46, 1985.
3. P. Bohannon, J. Freire, P. Roy, and J. Simeon. From xml schema to relations: A cost-based approach to xml storage. In *ICDE*, 2002.
4. A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with stored. In *SIGMOD*, pages 431–442, 1999.
5. I. Dinur and S. Safra. The importance of being biased. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 33–42. ACM Press, 2002.
6. D. Florescu and D. Kossman. Storing and querying xml data using an rdbms. In *Data Engineering Bulletin*, volume 22, 1999.
7. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
8. C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
9. Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM (JACM)*, 27(4):633–655, 1980.
10. A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas. Efficient relational storage and retrieval of xml documents. *Lecture Notes in Computer Science*, 1997, 2001.
11. A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.
12. J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. Relational databases for querying xml documents: Limitations and opportunities. In *Proceedings of the VLDB Conference*, 1999.

A Horizontal partitioning and the *Grouping* problem

Though the *Grouping* problem may look very similar to the problem of horizontal partitioning of a relational table on a single disk, there are a few differences. While horizontal partitioning is usually done in physical database design, the *Grouping* problem deals with logical database design while automatically finding a good relational schema for the given XML schema. Moreover, in the XML context how we translate an XML query into a relational query plays an important role in the problem. For example, if we want only Africa entries from the InCategory relation, we will have to perform additional joins to get the results. The tree structure of an XML schema and presence of wild cards in XML queries also create situations where an arbitrary subset of the Item elements can be selected. On the other hand, in the horizontal partitioning context, the domain is usually ordered and queries either select a single partition or a range of partitions. So, the *Grouping* problem is actually a generalization of the horizontal partitioning problem. The latter corresponds to the *Grouping* problem when all the weights are 0. In this scenario, under metric *RelCount* the problem is MaxSNP-Hard, while under metric *RelSize* the problem is in P for the three translation algorithms considered in this paper.