# XML Data Integration using Fragment Join

Jian Gong, David W. Cheung, Nikos Mamoulis, and Ben Kao

Department of Computer Science, The University of Hong Kong
Pokfulam, Hong Kong, China
{jgong,dcheung,nikos,kao}@cs.hku.hk

**Abstract.** We study the problem of answering XML queries over multiple data sources under a schema-independent scenario where XML schemas and schema mappings are unavailable. We develop the fragment join operator — a general operator that merges two XML fragments based on their overlapping components. We formally define the operator and propose an efficient algorithm for implementing it. We define schema-independent query processing over multiple data sources and propose a novel framework to solve this problem. We provide theoretical analysis and experimental results that show that our approaches are both effective and efficient.

## 1 Introduction

Data integration allows global queries to be answered by data that is distributed among multiple heterogeneous data sources [1]. Through a unified query interface, global distributed queries are processed as if they were done on a single integrated data source. To achieve data integration, a schema mapping is often used, which consists of a set of mapping rules that define the semantic relationship between the global schema and the local schemas (at the data sources). In these systems, such as Clio [2], processing a global query typically involves two steps: query rewriting, and data merging.

While much work has been done on query rewriting, very little has been done on data merging. In most existing approaches, data merging is mostly an *ad hoc* computation — a special data merging routine is custom-coded for each mapping rule. This approach leads to inflexible system design. In this paper we propose a schema independent framework that allows data merging be processed without referring to any specific schema mapping rules.

Let us illustrate our idea by an example. Figure 1(a) shows two XML documents taken from UA Cinema website and IMDB website, respectively. Both UA and IMDB contain the *title* and the *director* of each *movie*. In addition, UA contains *venue* and *price*, while IMDB contains the movie's *reviews*. Consider a user who wants to find out the *title*, *director*, *price*, and *review* for each *movie*. This is expressed by the twig pattern query shown in Figure 1(b). Note that neither UA nor IMDB can answer the query alone because UA lacks *reviews* and IMDB lacks *pricing* information. The (global) query thus has to be broken into two query fragments, one for each site. The returned results from the two sites should then be merged based on their common components. Figure 1(c) shows an example of the query result. Our goal is to answer such twig pattern queries in a schema-independent fashion where mapping rules are not needed.

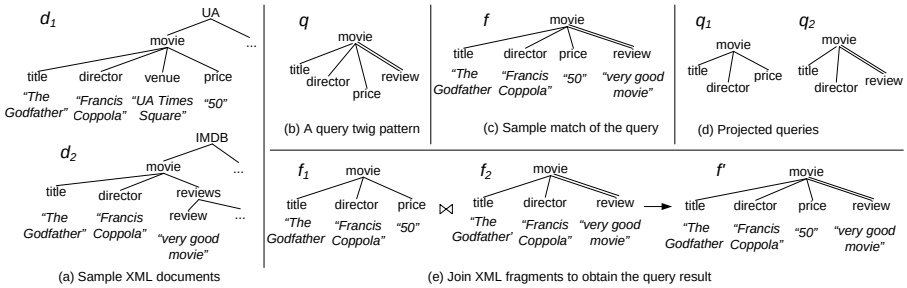



**Fig. 1.** Query on sample XML documents and the results.

In our approach, we join data fragments based on their overlapping content in order to answer queries. For example, we first *project* the global query on the two XML documents and obtain two local queries (Figure 1(d)). Then, we retrieve XML fragments $m(t, d, p)$ from UA and $m(t, d, r)$ from IMDB. Afterward, we *join* these fragments based on their overlapping parts, which are title ($t$) and director ($d$) (Figure 1(e)).

## 2 Preliminaries

An XML document $D$ is a rooted, node-labeled tree $D = \langle N, E, r \rangle$, wherein $N$ is a node set, $E \subseteq N \times N$ is an edge set, and $r \in N$ is the root node. Each node in an XML document has a label and may contain some text. The *vocabulary* of an XML document $d$, denoted by $v(d)$, is the set of distinct node labels of $d$.

**Definition 1.** (XML FRAGMENT) *An XML fragment $f$ is an edge-labeled XML document, where each edge is labeled by either "/" (parent-child edge) or "//" (ancestor-descendant edge). An XML fragment $f$ is a fragment of an XML document $d$, denoted as $f \sqsubseteq d$, if there exists an injective mapping $\lambda : f.N \rightarrow d.N$, such that: (i) $\forall n \in f.N$, $n = \lambda(n)$, and (ii) $\forall e(n_1, n_2) \in f.E$ labeled as "/" (resp., "//"), $\lambda(n_1)$ is the parent (resp., ancestor) of $\lambda(n_2)$.*

**Definition 2.** (TWIG PATTERN AND MATCH) *A twig pattern is an XML fragment, where the text content of the nodes is disregarded. A fragment $f$ is a* **match** *to a twig pattern $q$, denoted as $f \vdash q$ , if there exists a mapping $\gamma : q.N \rightarrow f.N$, such that the node labels and edges of $q$ are preserved in $f$. A fragment $f_1$ is contained in another fragment $f_2$, denoted as $f_1 \preceq f_2$, if all the nodes and edges of $f_1$ are contained in $f_2$.*

**Definition 3.** (PROJECTION) *Given a fragment $f$ and a vocabulary $v(d)$ of a document $d$, the projection of $f$ on $v(d)$, denoted as $\rho_{v(d)}(f)$, is obtained by removing from $f$ all the nodes whose labels are not in $v(d)$ and the corresponding connecting edges.*

## 3 The fragment join operator

**Definition 4.** (FRAGMENT JOIN) *Given a set of of fragments $f_1$, ..., $f_n$ ($n \geq 2$), a fragment $f$ is a join of $f_1$, ..., $f_n$, denoted as $(f_1, ..., f_n) \overset{\bowtie}{\rightarrow} f$, if $\exists f'_1 \preceq f, ..., f'_n \preceq f$,*

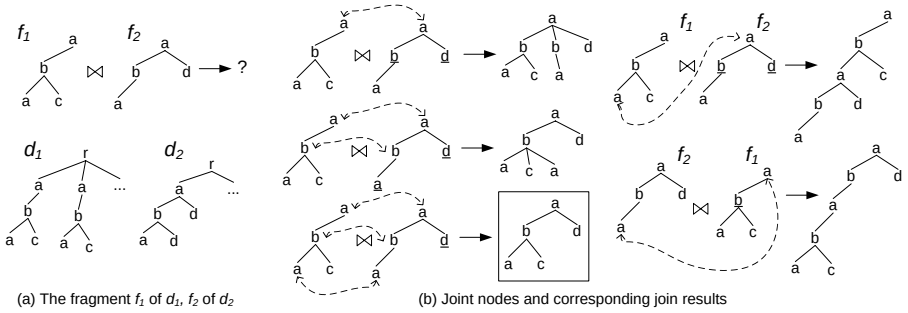



**Fig. 2.** XML fragment join on different joint sub-trees.

*such that: 1)* $f'_i = f_i$, $1 \leq i \leq n$, *2)* $\forall n \in f.N$, $n \in f'_1.N \cup ... \cup f'_n.N$, *and 3)* $\forall e \in f.E$, $e \in f'_1.E \cup ... \cup f'_n.E$.

*In addition, the join set of* $f_1$, ..., $f_n$ *is a set of fragments* $F = \{f | (f_1, ..., f_n) \overset{\bowtie}{\rightarrow} f\}$, *denoted as* $(f_1, ..., f_n) \overset{\bowtie}{\Rightarrow} F$.

**Definition 5.** (JOINT SUB-TREE) *Given two fragments* $f_1$ *and* $f_2$, *a subtree* $js$ *is a joint sub-tree of* $f_1$ *and* $f_2$ *if (1)* $js \preceq f_1$, $js \preceq f_2$, *(2) the root of* $js$ *= the root of* $f_2$.

Figure 2(b) shows the five results of the fragment join between $f_1$ and $f_2$ shown in Figure 2(a). Each of these results is based on a joint sub-tree, whose nodes are pointed by double-arrowed dashed lines in the two fragments.

We propose Algorithm 1 for evaluating the fragment join of two fragments $f_1$ and $f_2$. For example, consider the first join result shown in Figure 2(b). The joint-subtree for this join result consists of a lone node $a$. The *boundary nodes* are the children of the root node $a$ in $f_2$, which are labeled $b$ and $d$ (underlined). The subtrees of these boundary nodes are attached to the matching node $a$ in $f_1$ forming the join result.

## 4 Schema-independent, query-based data integration

Our research problem is formally stated as following: *given XML documents* $d_1$ *and* $d_2$, *and a twig pattern query* $q$, *compute* $F = \{f | f \vdash q; (f_1, f_2) \overset{\bowtie}{\rightarrow} f; f_1 \sqsubseteq d_1; f_2 \sqsubseteq d_2\}$. Our approach to solve this problem consists of the following phases.

**Projection.** The twig query $q$ is rewritten into local queries $q_1 = \rho_{v(d_1)}(q)$ and $q_2 = \rho_{v(d_2)}(q)$ using the *project* operator (Section 2). We then apply the fragment join operator on $q_1$ and $q_2$ to find a joint sub-tree $js$ for which the join result is $q$.

**Matching.** Two sets of fragments $F_1$ and $F_2$ are returned, which contains all matches to the local query $q_1$ in $d_1$ and all matches to the local query $q_2$ in $d_2$, respectively [1] .

**Join.** For each pair of fragments $(f_1, f_2) \in F_1 \times F_2$, we compute the fragment join of $f_1$ and $f_2$ using the joint-subtree obtained in the *projection* phase. The join results are returned as the query's answer.

[1] We thank the authors of [3] for providing us with the implementation of TwigList, used as a module for evaluating twig queries in our work.

**Algorithm 1** The join evaluation algorithm

**Input:** XML fragments $f_1$ and $f_2$
**Output:** a set of XML fragments $F$, with the join sub-trees used for each $f \in F$

1: $JS \leftarrow enumerateJointSubtrees(f_1, f_2)$
2: **for all** $js \in JS$ **do**
3: $f \leftarrow join(f_1, f_2, js)$
4: output $(f, js)$
5: **end for**
6: repeat 1-6 with $f_1$ and $f_2$ exchanged, if necessary

**function** $join(f_1, f_2, js)$

1: $f \leftarrow copy(f_1)$
2: **for all** $x \in js.N$ **do**
3: let $x_1$, $x_2$ be the corresponding nodes of $x$ in $f_1$ and $f_2$, respectively
4: **for all** $x_2$'s child $c$ **do**
5: **if** $c \notin js.N$ **then**
6: $sf \leftarrow constructFragment(f_2, c)$
7: $addChild(f_1, x_1, sf)$
8: **end if**
9: **end for**
10: **end for**
11: return $f$

Figure 3 illustrates our approach (the found joint sub-tree contains the underlined nodes). We note that projecting a global query onto local sources so that one single local query is applied to each source may not be sufficient to retrieve the complete set of query results. For example, consider again query $q$ in Figure 3. We observe that joining $q_{11}$, a sub-twig pattern of $q_1$ containing nodes $b$ and $c$ and the edge between them, with $q_2$ also gives us $q$ (using the joint sub-tree $b$) . Therefore, in order to ensure that all valid query results are found, we should consider all pairs of sub-twig patterns of $q_1$ and $q_2$ that can form $q$.

**Definition 6.** (RECOVERABILITY) *Given a twig pattern q, a pair of twig patterns ($q_i$, $q_j$) is recoverable for q, denoted as* $(q_i, q_j) \overset{r}{\rightsquigarrow} q$, *if* $(q_i, q_j) \overset{\bowtie}{\rightarrow} q$ *using some joint sub-tree js; else, ($q_i$, $q_j$) is non-recoverable for q, denoted as* $(q_i, q_j) \overset{r}{\not\rightsquigarrow} q$.

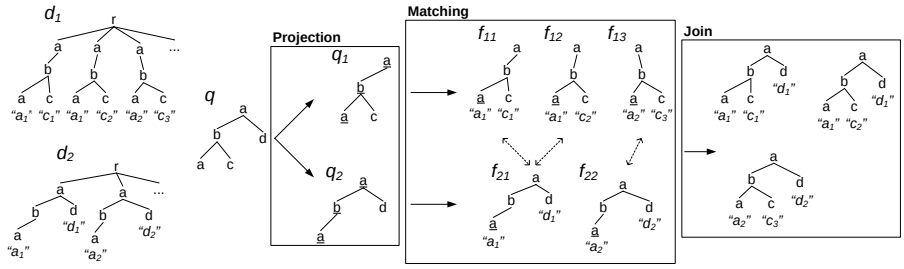
We add two more schema-level phases to the Projection-Matching-Join framework, in order to ensure completeness of the query results.

**Decomposition.** After the projection phase in which local queries $q_1$ and $q_2$ are derived, the decomposition phase returns: $Q_1 = \{q_i | q_i \preceq q_1\}$, and $Q_2 = \{q_j | q_j \preceq q_2\}$.

**Recoverability checking.** After the decomposition phase, this phase returns: $\{(q_i, q_j) | (q_i, q_j) \in Q_1 \times Q_2 \wedge (q_i, q_j) \overset{r}{\rightsquigarrow} q\}$.

## 5 Experimental evaluation and conclusion

We use DBLP and CiteSeer datasets in our experiments. The raw CiteSeer data are in plain text BibTeX format. We converted them into an XML file having similar schema




(a) Two XML documents (b) The twig pattern query and *Projection-Matching-Join* query answering process

**Fig. 3.** Query answering from multiple data sources: projection, matching, and join.

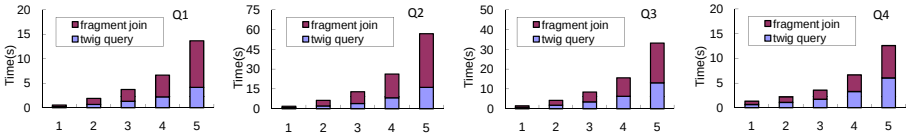



**Fig. 4.** Overall performance of PDRMJ for all queries and datasets.

to that of DBLP data. The size of Citeseer dataset is 15MB. We randomly sample the original DBLP (130MB) dataset to extract the publication records and attributes, and obtain five DBLP datasets, whose sizes are: 1MB, 10MB, 20MB, 40MB, and 80MB, respectively. Thus, we have five pairs of datasets used for the queries, each consisting of the Citeseer dataset plus one of the sampled DBLP datasets.

We manually created four test twig pattern queries, named Q1-Q4, each of which queries on a set of atrributes of papers, such as *title*. All these queries can only be answered using both DBLP and Citeseer datasets (but not one of the two datasets alone) by fragment join in our framework.

The overall performance of our complete, optimized approach (PDRMJ) is tested in Figure 4 for all queries Q1-Q7 on all datasets. The overall response time is broken down to two parts: (i) the time spent by all sub-twig pattern queries issued against the different sources, and (ii) the time spent by the fragment joins. We observe that the performance for all queries scales roughly linearly to the size of the DBLP dataset (recall that the size of the CiteSeer dataset is fixed). In addition, nearly half of the cost is due to the twig pattern queries against the sources.

In conclusion, we developed a fragment join operator for query-based data integration from multiple sources. We studied the problem of schema-independent data integration based on this operator. We conducted experiments to show the effectiveness of our approaches.

## References

1. Lenzerini, M.: Data integration: a theoretical perspective. In: PODS. (2002)
2. Yu, C., Popa, L.: Constraint-based XML query rewriting for data integration. In: SIGMOD. (2004)
3. Qin, L., Yu, J.X., Ding, B.: TwigList: make twig pattern matching fast. In: DASFFA. (2007)