Merging Interface Schemas on the Deep Web via Clustering Aggregation

Wensheng Wu, AnHai Doan
University of Illinois, Urbana, USA

Clement Yu
University of Illinois, Chicago, USA

Abstract

We consider the problem of integrating a large number of interface schemas over the Deep Web, The scale of the problem and the diversity of the sources present serious challenges to the conventional manual or rule-based approaches to schema integration. To address these challenges, we propose a novel formulation of schema integration as an optimization problem, with the objective of maximally satisfying the constraints given by individual schemas. Since the optimization problem can be shown to be NP-complete, we develop a novel approximation algorithm LMax, which builds the unified schema via recursive applications of clustering aggregation. We further extend LMax to handle the irregularities frequently occurring among the interface schemas. Extensive evaluation on real-world data sets shows the effectiveness of our approach.

1. Introduction

The *Deep Web* consists of a large number of Web databases whose contents are hidden behind their query interfaces. Virtual data integration over Deep Web sources is an emerging research problem which has received great attention (e.g. [2, 4, 5, 6]). The challenges of the problem arise largely from two aspects: (1) *scale*: there are typically a large number of Web databases in any domain of interest; and (2) *diversity*: these databases often vary greatly in their structure, coverage, vocabulary, and querying capabilities.

As an important step towards the integration of Web databases, we consider the problem of integrating their query interfaces. Query interface to a Web database is typically *structured*, containing a set of query attributes which are grouped and ordered based on their relative semantics. The structure of the interface can be naturally represented with a hierarchical schema such as *ordered tree* [6]. To illustrate, Figure 1(a) shows a query interface to an airfare database and Figure 1(b) shows its schema as an ordered tree.

The integration of interface schemas is typically accomplished in two steps: *schema matching*, which identifies *semantic* correspondences among interface attributes; and *schema merging*, which constructs a *unified* schema given the discovered mappings of the attributes. Such a unified schema should encompass all *unique* attributes over the given set of interfaces, and should be structurally and semantically well-formed. For example, Figure 1(c) shows schema S_v for a different airfare query interface, where attribute marked with x' matches with attribute x in the schema S_u of Figure 1(b). Figure 1(d) shows a unified schema which integrates the schemas S_u and S_v.

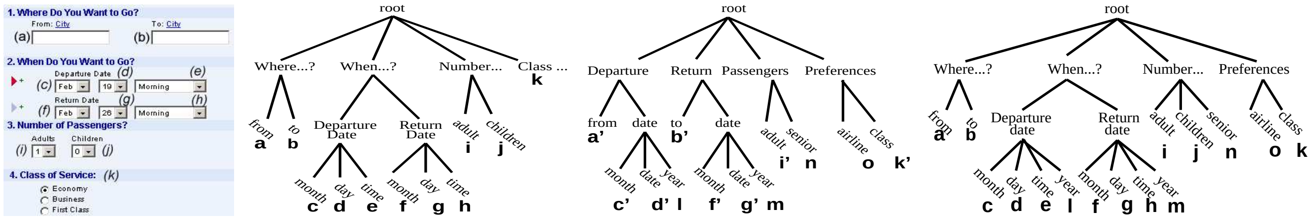
While interface schema matching has been well studied [4, 5, 6], the problem of merging interface schemas has received relatively little attention. As discussed above, the interface schemas are greatly diversified due to the autonomous nature of the sources. As a result, *structural conflicts*, as exemplified by the fact that different interfaces may represent a different set of attributes and may organize the attributes in quite different ways, are prevalent over the interfaces. For example, while schema S_u arranges the attributes by location and date, schema S_v groups them on departure and returning. The resolution of structural conflicts over a large number of interface schemas thus poses serious challenges that call for a novel scalable solution.

In this paper, we present a first *systematic* study on the problem of merging a *large number* of interface schemas. We propose a novel formulation of schema integration as an *optimization* problem (Section 2), where each interface schema in essence expresses certain constraints on the unified schema and the goal is to construct a unified schema such that these constraints are maximally satisfied. Since the optimization problem can be shown to be NP-complete, we propose a novel approximation algorithm LMax, based on recursive applications of *clustering aggregation* [3] (Section 3). We further extend LMax to cope with the irregularities prevalent over interface schemas (Section 4). Finally, we present the experimental results (Section 5). The extension to LMax for producing ordered schemas is given in the full version of the paper.

2. Schema Integration as an Optimization Problem

First, we formally define interface schemas, constraints, and the schema integration problem.

Definition 1: (**Interface schema**) We view each interface schema as an *ordered tree* of elements. Each leaf element corresponds to an attribute on the interface. Each internal



(a) An airfare query interface Q

(b) S_u: the schema of Q

(c) S_v: the schema of a different airfare interface

(d) A schema unifying S_u and S_v

FIGURE 1: Examples of query interface, schema, and unified schema

element has an ordered set s ($|s| > 1$) of sub-elements, each of which may be either a leaf element or an internal element. Sub-elements are ordered by the sequence their corresponding attributes (if leaf elements) or groups of attributes (if internal elements) appear on the interface. ■

For example, Figure 1(b) shows the schema S_u for the interface in Figure 1(a). Elements in the schema are annotated with labels from the interface. In the following, we will also use parenthesis notation to represent the schemas. For example, schema S_u may be represented as ((a, b) ((c, d, e) (f, g, h)) (i, j) k).

As discussed earlier, each of the interface schemas to be integrated essentially expresses certain preferences over the unified schema. These preferences can be encoded in two types of constraints: (a) *structural constraints*, which are expressed over the ancestor-descendant relationships on the lowest common ancestor (LCA) of attributes and thus restrict the structure of the unified schema; and (b) *precedence constraints*, which are expressed over the sequence of attributes and thus restrict the ordering of the elements in the unified schema. We now formally define these constraints.

Definition 2: (**Structural constraint**) Consider a schema S and denote the lowest common ancestor of two attributes x and y in S as LCA(x, y). Consider three attributes x, y and z in S. We say that there exists a *structural constraint* in form of $(x, y)z$ from S, if $\text{LCA}(x, y) < \text{LCA}(x, z)$ and $\text{LCA}(x, y) < \text{LCA}(y, z)$, where $n_1 < n_2$ denotes that element n_1 is a proper descendant of element n_2. ■

Example 1: $(a, b)c$ is a structural constraint from the schema S_u. Intuitively, it indicates, according to S_u, attributes a and b (both on the location of flight) are more closely related than either to c (which is on the date of flight). ■

Given a set of structural constraints without conflicts, [1] gives a polynomial time algorithm to construct a tree which satisfies all the given constraints. But when there are conflicts in the given structural constraints, the algorithm in [1] can not be applied. In our integration problem, conflicts are prevalent. For example, $(a', c')b'$ from S_v conflicts with $(a, b)c$ from S_u. In general, constraint $(x, y)z$ conflicts with $(x, z)y$ and $(y, z)x$. Given two conflicting constraints, it is impossible to find a unified schema which satisfies both constraints.

Definition 3: (**Precedence constraint**) Consider a schema S and a sequence of attributes, denoted as q_s, obtained from a *pre-order* traversal of S. We say that there exists a precedence constraint between two attributes x and y, denoted as $x \prec y$, from the schema S, if x appears *before* y in q_s. ■

Example 2: The sequence q_{s_u} for schema S_u is $<a, b, c, d, e, f, g, h, i, j, k>$. As such, $a \prec b$, $a \prec d$ are two precedence constraints from S_u. ■

Based on the above definitions, we cast the integration problem as an optimization problem:

Integration problem OPT: Given a set of interface schemas $\mathcal{S}$ with a set of *distinct* attributes A, find a unified schema G such that (1) G's leaf elements are attributes in A; (2) the number of *structural constraints* from the schemas in $\mathcal{S}$, which are satisfied by G, is maximized; and (3) the number of *precedence constraints* from the schemas in $\mathcal{S}$, which are satisfied by G, is maximized.

3. Approximating OPT via Clustering Aggregation

It is not difficult to prove that OPT is NP-complete. This section presents the algorithm LMax which gives an approximate solution to OPT. Essentially, LMax views the construction of a unified schema as a process of forming recursive partitions over a set of attributes such that *structural constraints* from the interface schemas are satisfied as much as possible. To illustrate, consider integrating schemas S_u and S_v. To start with, LMax is given a set A of 15 unique attributes (numbered a–o) over the two schemas. At the first iteration, LMax will create a root node r, form a partition over the attributes in A, and then create a list of children for r, each corresponding to a cluster in the partition. The same process is then recursively applied to each child of r, given the attributes in the cluster associated with that child.

We first introduce several necessary concepts.

Definition 4: (**Cluster, maximum cluster, and clustering**) Consider a schema S which contains a set of attributes

```
LMax($\mathcal{S}$, $A$, $r$)
Input: $\mathcal{S}$, a set of interface schemas;
    $A$, a set of distinct attributes over the schemas in $\mathcal{S}$.
Output: $r$, the root of a unified schema $G$

(1) if $A = \{a\}$ /* $A$ has only one attribute */
    $r \leftarrow$ NODE($a$)
(2) else if $A = \{a, b\}$
    $r_1 \leftarrow$ NODE($a$), $r_2 \leftarrow$ NODE($b$), $r \leftarrow$ NODE($r_1$, $r_2$)
(3) else /* $A$ has at least three attributes */
    (a) $\{C_{r_1}, C_{r_2}, ..., C_{r_k}\} \leftarrow$ PARTITION($A$, $\mathcal{S}$)
    (b) for each $C_{r_i}$, $1 \leq i \leq k$, do
        $\mathcal{S}' \leftarrow \mathcal{S}|C_{r_i}$
        LMax($\mathcal{S}'$, $C_{r_i}$, $r_i$)
    (c) $r \leftarrow$ NODE($r_1$, ..., $r_n$)
```

FIGURE 2: The LMax algorithm

A. For each node n in S, we define a *cluster* C_n as the set of attributes (i.e. leaf elements) in the sub-tree rooted at n. A cluster C is a *proper* cluster if $C \subset A$. A proper cluster is a *maximum cluster* if it is not a subset of any other proper clusters in S. The set of all maximum clusters in S forms a *clustering* over the attributes in A. ■

Example 3: The maximum clusters in S_u are $\{a, b\}$, $\{c, d, e, f, g, h\}$, $\{i, j\}$, $\{k\}$, each associated with a child of the root. ■

Definition 5: (**Restriction**) A *restriction* of a schema S on a set of attributes X, denoted as $S|X$, is a schema given by: (1) pruning all attributes in S which are not in X; (2) pruning the internal node if all its children are pruned; (3) replacing the internal node with only one child by its child.

A *restriction* of a set of schemas $\mathcal{S} = \{S_1, ..., S_n\}$ on X, denoted as $\mathcal{S}|X$, is a set of schemas $\{S_1|X, ..., S_n|X\}$. ■

Example 4: $S_u|\{a, b, c, d, k\}$ = ((a, b) (c, d) k). ■

Based on the above definitions, LMax algorithm is given in Figure 2. Note that NODE(a) creates a leaf node with the attribute a; and NODE(n_1, ..., n_k) creates an internal node with n_i's as the children.

Given a set of schemas $\mathcal{S} = \{S_1, S_2, ..., S_n\}$, and a set of unique attributes A over the schemas in $\mathcal{S}$, LMax builds a unified schema G as follows. If A has only one attribute a, it simply returns NODE(a) as the root of G. If A has only two attributes a and b, G is a tree with two leaves: NODE(a) and NODE(b). Otherwise, it first forms a partition $P = \{C_{r_1}, C_{r_2}, ..., C_{r_k}\}$ over the attributes in A. Then for each cluster C_{r_i}, it recursively creates a sub-tree rooted at r_i based on a set of restricted schemas $\mathcal{S}|C_{r_i}$. Finally, it returns NODE(r_1, r_2, ..., r_k) as the root of G.

We now describe the PARTITION function, the key component of the LMax algorithm, in detail.

PARTITION: PARTITION(A, $\mathcal{S}$) finds a partition over the attributes in A such that the structural constraints from the schemas in $\mathcal{S}$ are satisfied as much as possible.

Consider a partition P over A where P = $\{C_1$, C_2, ..., $C_k\}$. Consider further a schema $S \in \mathcal{S}$. Suppose that M = $\{C'_1$, C'_2, ..., $C'_l\}$ is a set of the maximum clusters in S. We observe that, to satisfy as many structural constraints from S as possible, P should be such that if two attributes x, y are in the same cluster $C_i \in M$, then both x and y should also be placed in the same cluster, say $C_j \in P$. Otherwise, all constraints of the form $(x, y)z$ will be violated. On the other hand, having $x, y \in C_j$ will satisfy *all* the constraints of the form $(x, y)z$ for some $z \notin C_i$. In other words, we may regard M as a clustering over the attributes in A (possibly with some missing attributes), and a good partition P should be such that it agrees with M on the cluster labels of the attributes.

Denote the set of such clusterings as $\mathcal{M}$ = $\{M_1$, ..., $M_n\}$, where M_i is the clustering given by the schema S_i. Since different schemas may give different clusterings, our problem is a problem of *clustering aggregation*: we seek a partition P such that P maximally agrees with the clusterings in $\mathcal{M}$. Unfortunately, clustering aggregation is also a NP-complete problem [3]. As such, PARTITION implements an approximation algorithm which can be regarded as a variant of the AGGLOMERATIVE algorithm in [3].

Given a set of attributes A and a set of clusterings $\mathcal{M} = \{M_1, ..., M_n\}$, PARTITION proceeds as follows. For every pair of attributes a and b in A, their *potential* of being in the same cluster, denoted as $p(a, b)$, is given by the number of clusterings in $\mathcal{M}$ which place a and b in the *same* cluster, subtracted by the number of clusterings which place a and b in *different* clusters. PARTITION starts by placing each attribute in A in a cluster by itself. It then repeatedly merges two clusters with the largest potential, where the potential of two clusters is given by the *group-average* of the potentials of the attributes in the two clusters. The merging process stops when no two clusters have a *positive* potential.

Example 5: Consider $\mathcal{S} = \{S_u, S_v, S_w\}$, where S_u and S_v are schemas in Figure 1(b) and 1(c) respectively, and S_w is homogeneous to S_u (with the same attributes and structure). Thus, $\mathcal{S}$ contains a set A of 15 unique attributes numbered from a to o. The first call to PARTITION with A and $\mathcal{S}$ returns $P = \{\{a, b\}, \{c, d, e, f, g, h, l, m\}, \{i, j, n\}, \{k, o\}\}$. By recursive applications of PARTITION on each cluster in P, LMax produces a unified schema shown in Figure 1(d). ■

4. Handling Irregular Interface Schemas

Compared to other types of schemas, e.g., schemas in relational databases, interface schemas are typically much less regular. In particular, we observe that the structure of some interfaces may be *implicit* in that the attributes may be simply listed one after another without explicit group delimiters. This poses challenges to schema extraction algorithms. As a result, the obtained schema may not fully capture the grouping relationships of attributes on the interface. The irregular schemas may greatly affect the performance of PARTITION which assumes that maximum clusters of each schema S indicate S's preferences on dividing attributes into groups.

To address this challenge, consider again a set of schemas $\mathcal{S}$, some of which may be irregular. A key observation is that we can exploit other schemas in $\mathcal{S}$ to identify the irregularities in the irregular schemas, assuming that not every schema in $\mathcal{S}$ is irregular. Specifically, consider a schema $S_i \in \mathcal{S}$. We observe that if attributes $a, b \in S_i$ appear in different maximum clusters of S_i, but both appear in the same maximum cluster of some other schema $S_j \in \mathcal{S}$, then it is very likely that the grouping relationship between a and b is implicit on the interface of S_i. Motivated by the above observation, we extend LMax based on the concept of *global maximum clusters* defined as follows.

Definition 6: (**Global maximum cluster**) Consider a set of schemas $\mathcal{S} = \{S_1, ..., S_n\}$, where each schema $S_i \in \mathcal{S}$ gives a set of maximum clusters $M_i = \{C_{i_1}, C_{i_2}, ..., C_{i_k}\}$. We say that a maximum cluster $C_{i_j} \in M_i$ is a *global maximum cluster* if it is not a proper subset of any maximum clusters of any other schemas in $\mathcal{S}$. ■

We denote the set of global maximum clusters over the schemas in $\mathcal{S}$ as $\mathcal{C}^{\mathcal{S}} = \{C_1^S, C_2^S, ..., C_m^S\}$, and denote the set of unique attributes in the schemas of $\mathcal{S}$ as A. (It is important to note that the clusters in $\mathcal{C}^{\mathcal{S}}$ may not form a partition over A, due to structural conflicts in the schemas of $\mathcal{S}$.)

Based on the above definition, we modify PARTITION in LMax. Recall that, given a set of attributes A and a set of clusterings $\mathcal{M} = \{M_1, ..., M_n\}$, where M_i is a set of maximum clusters obtained from schema $S_i \in \mathcal{S}$, PARTITION forms a partition over A via clustering aggregation.

The modified PARTITION consists of the following steps: (1) obtain $\mathcal{C}^{\mathcal{S}}$, the set of global maximum clusters; (2) transform $\mathcal{M}$ into a new set of clusterings $\mathcal{M}' = \{M'_1, ..., M'_n\}$, where M'_i is obtained from M_i by combining clusters in M_i, which are subsets of the same global maximum cluster in $\mathcal{C}^{\mathcal{S}}$, into one cluster; (3) perform the clustering aggregation with $\mathcal{M}'$ instead of $\mathcal{M}$.

The LMax algorithm with the new PARTITION is denoted as GMax.

5. Empirical Evaluation

We have evaluated both the LMax and GMax algorithms on a real-world data set over varied domains. The goal of the experiments was to examine if the produced unified schemas are semantically well-formed, and to compare the performance of the two algorithms.

For all experiments, we used a data set which contains a total of 100 interface schemas extracted from the query interfaces to Web databases in five domains: airfare, auto, book, job, and real estate, with 20 schemas for each domain. Table 1 gives the statistics of the data set.

Domain	Leaf Nodes			Internal Nodes			Depth		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Airfare	5	15	10.7	1	7	5.1	2	5	3.6
Automobile	2	10	5.1	1	4	1.7	2	3	2.4
Book	2	10	5.4	1	2	1.3	2	3	2.3
Job	3	7	4.6	1	2	1.1	2	3	2.1
Real Estate	3	14	6.7	1	6	2.4	2	4	2.7

TABLE 1: Domains and statistics of the data set

An objective measure on the quality of a produced unified schema is to compare the unified schema with the *optimal* unified schema. But since finding optimal schemas is computationally expensive, we use an alternative measure *PerSC*, which is the percentage of *strong* structural constraints from the given set of interface schemas satisfied by the unified schema. A structural constraint $(x, y)z$ is a strong constraint if it appears more often in the given set of interface schemas than its conflicting constraints, i.e., $(x, z)y$ and $(y, z)x$. Intuitively, since conflicting constraints from different schemas can not be satisfied simultaneously, we expect that the optimal schema should satisfy the *strong* constraints.

Alg.	Airfare	Auto	Book	Job	Real Est.	**Average**
LMax	73.6	74.0	91.8	73.7	63.6	75.3
GMax	89.9	95.6	91.8	89.5	89.0	91.2

TABLE 2: The performance of LMax vs. GMax

Table 2 shows the performance of LMax and GMax on the data set, measured by their *PerSC* scores. We observe that the *PerSC* scores of LMax range from 63.6% in the real estate domain to 91.8% in the book domain. We further observe that GMax improves the performance significantly over four domains, with a 15.8% increase in the job domain and as high as 25.4% increase in the real estate domain. These indicate the prevalence of irregularities in the interface schemas. Overall, the average *PreSC* score increases from 75.3% to 91.2%. This indicates the effectiveness of GMax in handling the irregular interface schemas.

References

[1] A. Aho, Y. Sagiv, T. Szymanski, and J. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM*, 10(3).

[2] L. Barbosa and J. Freire. Searching for hidden-web databases. In *WebDB*, 2005.

[3] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. In *ICDE*, 2005.

[4] B. He and K. Chang. Statistical schema matching across Web query interfaces. In *Proc. of SIGMOD*, 2003.

[5] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: an automatic integrator of web search interfaces for e-commerce. In *VLDB*, 2003.

[6] W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the Deep Web. In *SIGMOD*, 2004.