Data & Knowledge Engineering 86 (2013) 38–60

Contents lists available at SciVerse ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Multi-objective optimization integration of query interfaces for the Deep Web based on attribute constraints

Yanni Li [a,b,*], Yuping Wang [a], Peng Jiang [c], Zhensong Zhang [d]

[a] School of Computer Science and Technology, Xidian University, No. 2 South Taibai Road, Xi'an, Shaanxi 710071, PR China
[b] School of Software, Xidian University, No. 2 South Taibai Road, Xi'an, Shaanxi 710071, PR China
[c] Institute of Computing Technology Chinese Academy of Sciences, No. 6 Kexueyuan South Road Zhongguancun, Haidian District Beijing 100190, PR China
[d] Institute of Software Chinese Academy of Sciences, No. 4 Kexueyuan South Road Zhongguancun, Haidian District Beijing 100190, PR China

ARTICLE INFO

Article history:
Received 1 September 2011
Received in revised form 25 December 2012
Accepted 7 January 2013
Available online 16 January 2013

Keywords:
Domain-Specific Deep Web Data Integration Systems (DWDIS)
Algorithm
Integration of query interface
Attribute constraint matrix

ABSTRACT

In order to query and retrieve the rich and useful information hidden in the Deep Web efficiently, extensive research on domain-specific Deep Web Data Integration Systems (DWDIS) has been carried out in recent years. In DWDIS, large-scale automatic integration of query interfaces of domain-specific Web Databases (WDBs) remains a serious challenge due to the scale of the problem and the great diversity of the WDBs' query interfaces. To address this challenge, in this paper, we first give a definition of the constraint matrix which can accurately describe three types of constraints (hierarchical constraints, group constraints and precedence constraints) and the strengths of attributes of a query interface, and then prove that the schema tree of the query interface corresponds to only one constraint matrix, and vice versa. Furthermore, we transform the problem of integrating domain-specific query interfaces into a problem of integrating the constraint matrices and set up a multi-objective optimization problem model. To effectively solve the optimization model, some strategies to extend and merge the constraint matrices are designed. A method for automatically detecting and filtering abnormal data (noises) in the query interfaces is also proposed. More importantly, a novel and efficient algorithm applicable to large-scale automatic integration of domain-specific query interfaces is developed. Finally, the proposed algorithm is evaluated by experiments on the real query interface data set. Our theoretical analysis and experimental results show that the proposed algorithm outperforms existing state-of-the-art integration algorithms of domain-specific query interfaces.

1. Introduction

Nowadays, the Web has evolved into a data-rich repository containing significant structured contents [42,44]. These contents reside mainly in Web databases (WDBs) that are also referred to as the Deep Web. Recent surveys estimate that there are millions of such sources available [1–5,30,43]. Every WDB provides a query interface for users to access, which is the sole entry point to the WDB, such as the job portals or the search entry for cheap airline tickets. A query interface for booking airline tickets is shown in Fig. 1(a). In order to obtain the contents of the WDBs, a user has to submit structured queries by filling valid input values in Web query interfaces.

With the explosively increasing amount of domain-specific WDBs, it is unrealistic to expect users to explore each WDB individually. It is necessary to construct a domain-specific Deep Web Data Integration System (DWDIS) in which only a unified

* Corresponding author at: School of Computer Science and Technology, Xidian University, No. 2 South Taibai Road, Xi'an, Shaanxi 710071, PR China. Tel.: +86 2988202457.
E-mail addresses: yannili@mail.xidian.edu.cn (Y. Li), ywang@xidian.edu.cn (Y. Wang), jiangpeng1988@gmail.com (P. Jiang), zhensongzhang@gmail.com (Z. Zhang).

0169-023X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.datak.2013.01.003

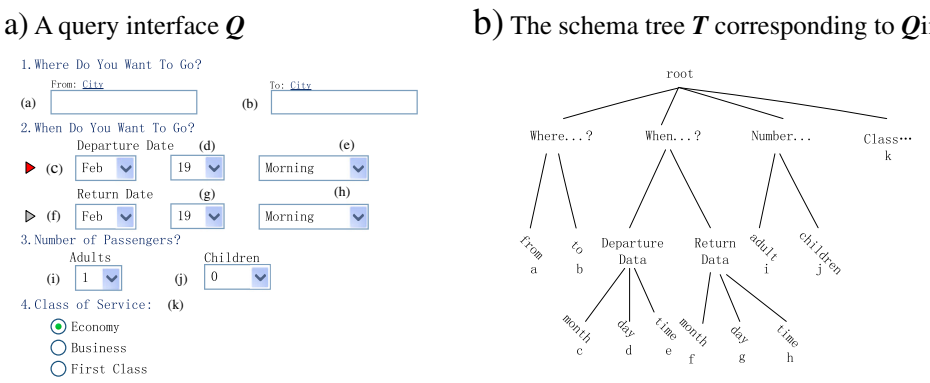


Fig. 1. A query interface and its schema tree in an airfare domain.

query interface is provided to users so that they can easily access to the large amount of data hidden in the WDBs. To achieve this purpose, DWDIS should include the following functions [3,20,21,36,38,39,46]: 1) identifying all WDBs; 2) clustering/classifying domain-specific WDBs; 3) matching schemas across a set of query interfaces of the domain-specific WDBs; 4) constructing an integrated query interface, or a unified query interface, for the domain-specific WDBs' query interfaces; 5) translating the query of the unified query interface to that of other individual query interfaces of the domain-specific WDBs; and 6) cleaning and integrating the query results from the domain-specific WDBs and then returning the integrated results to the user.

Given a set of query interfaces in domain-specific WDBs, the integration of the query interfaces is to automatically construct a unified query interface which contains all (or, alternatively, the most significant) distinct attributes of all the query interfaces and reasonably arrange them (called a well-designed query interface, that is, the unified query interface should preserve as much the structural constraints of all or most of the query interfaces as possible) so as to facilitate users' uniform access to the WDBs. Since the unified query interface is the only entry point to the DWDIS for users, the integration of the domain-specific query interfaces is one of the most fundamental and crucial components in the DWDIS. Typically, the organization of attributes within a query interface always has a meaningful purpose and enforces certain structural constraints (such as hierarchical constraints, group constraints and precedence constraints), which is one of the most critical factors in a well-designed query interface. Moreover, different query interfaces may represent a different set of attributes and may enforce different attribute structural constraints due to the autonomous nature of the query interfaces. As a result, the conflicts of attribute structural constrains will often occur. For example, in the airfare domain, a query interface puts attributes the departure day and the return day into a group, while another query interface puts them into two different groups, respectively. Therefore, large-scale automatic integration of autonomous query interfaces of domain-specific WDBs is a serious challenge due to the large scale and great diversity of the WDBs' query interfaces [3,11,22,23,36,39,41,48].

In order to better describe the various structural constraints between attributes of a query interface, the query interface can be naturally represented by a hierarchical schema such as ordered trees [6,15]. Therefore, integrating query interfaces of domain-specific WDBs is virtually merging/integrating these query interface schemas, which mainly includes following steps corresponding to the aforementioned functions 3) and 4) in DWDIS. First, domain-specific query interfaces from individual Web pages are extracted along with the attributes and their parent–child or group relationships in the query interfaces [11,13,22,23,29,31,37]. Second, the semantic relationships between the attributes in different query interfaces are computed, which is referred to as the schema matching of query interfaces [6,8–10,17]. Third, the query interfaces are integrated into a unified interface. In this study, we concentrate mainly on how to automatically and efficiently integrate a huge number of heterogeneous query interfaces of domain-specific WDBs and construct a well-designed unified query interface, which corresponds to the aforementioned third step, noting that the semantic mappings have been obtained. Although attribute value merging and format integration belong to this step, they will not be discussed here due to space limitation. A comprehensive treatment of the subject can be found in [41].

Although schema matching across a set of query interfaces on the domain-specific WDBs is a well-studied problem [14,18,19,32–34,40,45], few studies on the query interfaces integrating [7,12,26,35] and its relevant problems [16,24,25,27,47] have been carried out. Moreover, the previous work on this issue has the following limitations: first, the non-hierarchical query interface modeling proposed in [11,22,23,41] cannot reflect the rich structural constraint information, e.g., parent–child or group relationships between attributes of a query interface, without which it may yield an ill-designed unified query interface; second, for the existing hierarchical query interface modeling [7,12,15,31], there has been no metric which can accurately describe and quantify the various constraints among attributes of a query interface; third, the current state-of-the-art query interface integrating algorithms [7,12,21] remain to be improved in the aspects such as simplification and efficiency of the algorithms and properly modeling; fourth, due to the difficulties in prediction and evaluation on the quality of a unified query interface, none of the current integrating methods sets up a problem evaluation model which can measure the quality of the integration, or analyze the time complexity.

To overcome the defects of the existing integration methods, a deep and systematic study on integrating query interfaces has been carried out and a novel and efficient algorithm has been developed for integrating domain-specific query interfaces. The main work of the paper is summarized as follows:

1) To integrate query interfaces efficiently, a hierarchically ordered tree called the schema tree for each query interface is set up, and a one-to-one correspondence (mapping) between various attribute constraints and the schema tree of a query interface is identified;
2) A quantitative metric called the attribute constraint matrix is given, which is applicable to various attribute constraints of a query interface;
3) The query interfaces integrating problem is transformed into a multi-objective optimization problem. Based on this proposed multi-objective optimization model, some operations on the attribute constraint matrices are presented, and the mechanism of automatically detecting and filtering out abnormal (noises) data in the query interfaces to be integrated is set up, and then a novel and efficient algorithm applicable to large-scale automatic integration of the query interfaces is proposed by just utilizing the 1:1 schema match result without conflicts;
4) The problem evaluation model is proposed, which can measure the integration quality and the time complexity.
5) Simulation experiments in multiple domains from real query interfaces data set have been conducted, and both the theoretical analysis and the experimental results demonstrate the effectiveness and efficiency of the proposed model and algorithm.

The rest of this paper is organized as follows. Section 2 describes the method to model a query interface into a schema tree and gives the definition of the constraint matrix of attributes of a query interface over a WDB first, and then constructs and proves a one-to-one correspondence between the attribute constraint matrix of a query interface and its schema tree. Section 3 presents the model of transforming the integration of query interfaces in domain-specific WDBs into a multi-objective optimization problem. Section 4 discusses the proposed new algorithm for the transformed problem in detail, and detects and filters out noises in integrating query interfaces. Section 5 gives the simulation experiments on the real query interfaces data set and the analysis of the simulation results. Section 6 makes the comparisons with existing algorithms. The conclusions are made in Section 7.

2. The schema tree and the attribute constraint matrix of a query interface on the Deep Web

As discussed earlier, our goal is to automatically and efficiently construct a well-designed unified query interface over a large number of heterogeneous query interfaces of domain-specific WDBs. Therefore, the premises for achieving this end are effectively modeling the query interfaces, and accurately identifying and quantifying various structural constraints between attributes of the query interfaces.

In this section, firstly, a general hierarchical query interface modeling method is described. Secondly, an approach to classification of various structural constraints between attributes of the query interfaces is given. Finally, the fact that there exists a one to one correspondence between a unique attribute constraint matrix and each query interface has been revealed and proved, which can accurately describe various structural constraints and the strengths between attributes of the query interfaces.

2.1. The schema tree and various structural constrains between attributes of a query interface

For a domain-specific query interface, there are generally four types of basic building blocks: text input boxes, selection lists, radio buttons, and check boxes, which are generically called attributes/fields of the query interface. Each attribute usually has a label (descriptive text) associated with it, which becomes the attribute name. A text input box allows users to input whatever value they want while a selection list usually has a list of value (options), and a radio button/checkbox usually has a single value. Often multiple checkboxes or multiple radio buttons are used together to accomplish the same function as a selection list. For a formal definition of a query interface sees literature [41]. As an example, for the query interface in the airfare domain shown in Fig. 1 (a), there are two text input boxes labeled (a) and (b), eight selection lists labeled (c)–(j), and three radio buttons labeled "Class of Service", respectively.

To accurately express the logical structure of the attributes of a domain-specific query interface, the schema of a domain-specific query interface is first modeled as a hierarchically ordered tree of its attributes by using the method in literature [6,7,12,15]. This ordered tree is called the schema tree (or schema) of the query interface and can be defined as follows:

Definition 1. [Query interface schema tree]

Given a query interface, its schema tree $\boldsymbol{T}$ is a hierarchically ordered tree of elements, in which each leaf element (node) corresponds to an attribute in the query interface. Each internal element (node) is followed by an ordered set $\boldsymbol{S}$ ($|\boldsymbol{S}|>1$) of sub-elements, each of which may be either a leaf element or an internal element. The sub-elements are ordered according to the sequence of their appearance in the query interface.

Fig. 1(b) is an example which shows the schema tree $\boldsymbol{T}$ of the query interface $\boldsymbol{Q}$ in Fig. 1(a), where the leaf nodes in the schema tree $\boldsymbol{T}$ represent the attributes in the query interface (labeled a, b, c,…,and k, respectively).

By investigating and analyzing its schema tree of a domain-specific query interface, the following three types of structural constraints between attributes in the schema tree (attribute constraints for short) should be satisfied: 1) *hierarchical constraints*, which define the depth from each attribute (leave node) to the 0-th level root node in the schema tree, e.g., for the schema tree $\boldsymbol{T}$

shown in Fig. 1(b), "Where…" is the father of attributes labeled by *a* and *b*, the depth of attributes *a* and *b* is 2 in the schema tree *T*; 2) *group constraints*, which indicate a sibling relationship among attributes, and all members in each group have the same father node, e.g., for the schema tree in Fig. 1(b), the attributes *c*, *d*, and *e* obey a group constraint; and 3) *precedence constraints*, which show the order in which the attributes appear in the schema tree *T* from left to right. In this paper, if there is a precedence constraint between two attributes *x* and *y*, and *x* appears before *y* in $\boldsymbol{T}$ from left to right, it will be denoted as $x \prec y$ in a schema tree $\boldsymbol{T}$. For example, for the schema tree in Fig. 1(b), the attributes satisfy the precedence constraint: $a \prec b \prec \ldots \prec k$.

It is clear that each attribute's position (or attribute's structure) in its schema tree is mainly dominated by hierarchical constraints, followed by the group constraints, and then by the precedence constraints. Therefore, for an attribute in the schema tree, there exits the following priority relation on the above three types of constraints shown in Eq. (1):

$$\textit{hierarchical constraints} > \textit{group constraints} > \textit{precedence constraints} \tag{1}$$

where the notation $A > B$ means that *A* has the higher priority than *B*.

2.2. The attribute constraint matrix of a schema tree and its properties

For convenience, some definitions, e.g., the distance between two attributes and the attribute constraint matrix of the schema tree *T*, are first given.

Definition 2. [Distance between two attributes in a schema tree]

In a schema tree *T*, the distance between two different attributes, e.g., *i* and *j*, is the length of the shortest path between them, denoted as $dis(i, j)$, where the weight of every edge in the *T* is 1. Similarly, the distance between an attribute *i* and the *root* note is defined as the length of the shortest path from the *root* to the attribute *i*, called the depth/level of the attribute *i* and denoted as $depth(i)$. Note that *depth* (*root*) of root is defined as 0.

Definition 3. [Attribute constraint matrix *M* of a schema tree]

Suppose that a schema tree $\boldsymbol{T}$ has *n* leaf nodes corresponding to *n* attributes in a query interface *Q*, respectively, an *n*-order matrix (an $n \times n$ matrix) *M*, called the attribute constraint matrix (constraint matrix for short) of the schema tree *T*, is constructed if and only if its element in row *i* and column *j* is defined by

$$M[i,j] = \begin{cases} dis(i,j), & i \neq j \\ depth(i), & otherwise \end{cases} \quad (1 \le i,j \le n) \tag{2}$$

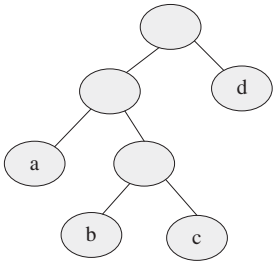
Obviously, the attribute constraint matrix *M* is a symmetric matrix. Note that matrix *M* can be obtained automatically based on the results of the schema extraction of a query interface.

Here is an example. The matrix shown in Fig. 2 (b) is the constraint matrix *M* of the schema tree *T* shown in Fig. 2 (a).

Based on Definition 3, the following Lemma can be easily obtained.

Lemma. The constraint matrix *M* of a schema tree *T* contains the information on all three types of constraints in the schema tree *T*, i.e., hierarchical constraints, group constraints, and precedence constraints. The value of each diagonal element of a matrix *M* describes the hierarchical constraints between a leaf node and root in the schema *T*, while the sequence of row (or column) elements in the matrix *M* indicates the precedence constraints, i.e., row 1 (column 1) to row *n* (column *n*) represents the leaf node 1 to leaf node *n* located from up (left) to down (right). Two different leaf nodes *i* and *j* are in one group (i.e., they satisfy a group constraint) if and only if $dis(i,j) = 2$.

a) Schema tree *T* of a query interface. b) Attribute constraint matrix *M* of the schema tree *T*.



	a	b	c	d
a	2	3	3	3
b	3	3	2	4
c	3	2	3	4
d	3	4	4	1

Fig. 2. Schema tree *T* and its corresponding attribute constraint matrix *M* of a query interface.

Obviously, each row (column) of the constraint matrix M corresponds to an attribute in a schema tree T, e.g., row i (column i) corresponds to the attribute i. Moreover, each schema tree corresponds to a unique constraint matrix, and vice versa, which can be described by the following theorem.

Theorem. There is a one-to-one correspondence between the constraint matrix M and the schema tree T of a query interface, that is, given a query interface's schema tree T, one and only one corresponding constraint matrix M can be obtained, and vice versa.

Proof.

1) Given a schema tree T of a query interface Q, according to Definition 3 in Subsection 2.2, only one corresponding constraint matrix M can be constructed.
2) Given an n-order constraint matrix M, it is used as the first temporary constraint matrix, each row (column) corresponds to a leaf node. Let $\{L_1,L_2,\ldots,L_n\}$ denote the set of leaf nodes of the schema tree T, and $L_1,L_2,\ldots,L_n$ corresponds to rows (columns) 1,2,…,n of the constraint matrix M, respectively. Then the schema tree T can be constructed by the following steps:

   a) Is the temporary matrix M a one-order (1×1) matrix? If not, go to b); else, go to g);
   b) Find all the maximum elements which must be greater than or equal to 2 on the diagonal of the matrix. These elements correspond to the depth of the deepest nodes in the schema tree. Store these elements in set G;
   c) Clustering those nodes in G with their distance being 2 into a cluster, that is, if the distance of the nodes in G is 2, the nodes will be in a cluster;
   d) Create a parent node for all nodes in each cluster to form a temporary schema tree (or forest), and use this parent node to represent the corresponding cluster (see the whole cluster as a node: parent node) temporarily;
   e) Remove all the rows and columns in each cluster of G from matrix M which are substituted by the parent node with its depth being 1 smaller than its child node. Then, update the distance between all the other nodes and the parent node in matrix M to form a new temporary matrix M;
   f) Go to a);
   g) End.

With steps *a*)–*g*), given a constraint matrix M, it is clear that one and only one schema tree T can be constructed. The proof is completed.

In fact, step *a*) to step *g*) in the proof of the above theorem makes up an algorithm called *MatrixToTree*, shown in Fig. 3, to generate the schema tree T from a given constraint matrix M. Note that the algorithm *MatrixToTree* employs a bottom-up approach to build its corresponding schema tree T based on a given constraint matrix M. The pseudo code of the algorithm *MatrixToTree* and its key component, i.e., the function *Update* for updating the distance between all elements in matrix M to form a new temporary matrix M, is shown in Fig. 3.

Here is an example to illustrate the algorithm *MatrixToTree* and its function *Update*. For a given constraint matrix M in Fig. 2 (b), in the first iteration of the algorithm *MatrixToTree*, the second and the third diagonal elements of the matrix M have the maximum value 3 ($G=\{b, c\}$, $Rest=\{a, d\}$), and the distance between nodes b and c is 2, so the nodes b and c satisfy the group constraint and should be in a cluster. A new node n_1 is created as the parent node of the nodes b and c, and the node n_1 and its children nodes b and c form a temporary schema tree T' shown in Fig. 4(a). Remove all the elements b and c (a cluster in G) from matrix M which are substituted by its parent node n_1 with its depth being 1 smaller than its children nodes b and c (i.e., 2). Then, update all the distance between all the other nodes and the parent node in matrix M by calling the function ***Update*** to form a new temporary constraint matrix M' shown in Fig. 4(b) ($New=\{n_1\}$, $Attri=\{a, n_1,d\}$).

Then, in the second iteration of the algorithm *MatrixToTree*, the deepest nodes a and n_1 (i.e., $G=\{a, n_1\}$, $Rest=\{d\}$) are selected and clustered. With the above steps of the algorithm *MatrixToTree* ($New=\{n_2\}$, $Attri=\{n_2,d\}$), another temporary schema tree T'' and its constraint matrix M'' are generated and shown in Fig. 5.

Finally, the remaining nodes n_2 and d in the temporary constraint matrix M'' are treated as the children of a parent node n_3 to form the final schema tree T, which is the same as the tree shown in Fig. 2 (a).

3. Multi-objective optimization model for query interfaces integration

3.1. Merging constraint matrices

According to Definitions 1 and 2, Lemma and Theorem in Section 2, it can be observed that: 1) Each query interface can be modeled by a schema tree; 2) each schema tree of a query interface can be represented by an only corresponding constraint matrix; and 3) a constraint matrix M of a schema tree T contains all information on the three types of constraints between the attributes of the query interface. Therefore, the problem of integrating query interfaces can be transformed into a problem of merging the constraint matrices of query interfaces.

Since the sizes of the constraint matrices for different query interfaces are usually different, attention needs to be paid in merging these matrices (some computations, e.g., addition, subtraction, multiplication and division, etc., for matrices with different sizes are

a) The pseudo code of algorithm ***MatrixToTree*** for constructing a schema tree T via a constraint matrix M.

Algorithm **MatrixToTree**(M)

Input: M: an n-order constraint matrix.

Output: T: a schema tree constructed by the M

```
1  m = M
2  create a leaf node for each attribute of M
3  while(m is not a 1×1 matrix)
4      select the collection of nodes whose value has a maximum value greater than or equal to 2
       in m's diagonal, and put them in set G;
5      select the collection of the remaining nodes, denoted as R;
6      divide G into n largest mutually disjointed attribute groups g1~gn and make sure the elements in each
       attribute group, say nodes i and j, belong to the same group if and only if m[i,j]==2, i≠j;
7      for each gk in { g1~gn } (1≤k≤n)
8          create a new node pk, which is the parent node of all the nodes in gk, i.e. add all nodes in gk
           into pk as its children, where each child of the node pk is denoted as pk.child;
9      endfor
10     Update(m, {p1~pn} , R)   //the function of updating all of the element values of the constraint matrix m
11 endwhile
12 return T
```

b) The pseudo code of function ***Update*** of algorithm ***MatrixToTree*** for updating the temporary constraint matrix ***m***.

Function **Update**(m, *New*, *Rest*)

Input: m: a temporary constraint matrix.

New: a set of newly created nodes, $\{p_1 \sim p_n\}$.

Rest: a set of remaining nodes

Output: m: an updated temporary constraint matrix

```
1  create a new matrix m′ whose attributes are Attri=New ∪ Rest;
2  for each node pi in Attri
3      for each node pj in Attri
4          pick up any two nodes ci and cj from pi.child and pj.child
5          if (pi and pj ∈ New)
6              if (pi==pj) m′[ pi,pi]= m[ci,cj]-1
7                  else m′[ pi,pj]= m[ci,cj]-2
8              else if (pi ∈ New, pj ∈ Rest)
9                  m′[ pi,pj]= m[ci,pi]-1
10             else if (pi ∈ Rest, pj ∈ New )
11                 m′[ pi,pj]= m[ pi,ci]-1
12             else if (pi and pj ∈ Rest)
13                 m′[ pi,pj]= m[ pi,pj]
14             else error
15         endfor
16 endfor
17 m = m′
18 return m
```

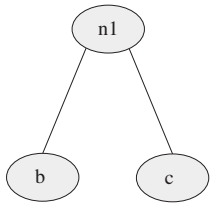
Fig. 3. Algorithm *MatrixToTree* for constructing a schema tree T via a constraint matrix M.

difficult or even nonsense). To facilitate the merging of matrices with different sizes, extended constraint matrices over a set of domain-specific query interfaces are defined as follows:

Definition 4. [Extended constraint matrices]

Suppose that there are n query interfaces' constraint matrices M_1, M_2, ..., and M_n, in which there are totally m different attributes a_1, a_2,..., and a_m. For any given sequence a_1, a_2,...,a_m of m attributes, the extended constraint matrix M'_k of M_k is an

a) Temporary schematree T'. b) Temporary constraint matrix M'.



	a	n1	d
a	2	2	3
n1	2	2	3
d	3	3	1

Fig. 4. Temporary schema tree T' and corresponding constraint matrix M' after removing nodes b and c which are substituted by its parent node n_1.

m-order symmetric matrix, where the rows or columns in M_k' are arranged in sequence $a_1, a_2,\ldots,a_m$, and its elements are defined by

$$M_k'[i,j] = \begin{cases} dis(i,j) & i,j \in M_k, i \neq j \\ depth(i) & i,j \in M_k, i = j \\ 0 & \text{otherwise} \end{cases} \quad (1 \le k \le n, 1 \le i,j \le m) \tag{3}$$

Obviously, M_k' can be obtained by making the same permutation on both rows and columns of M_k and adding zero rows and zero columns to M_k such that the sequence of rows (columns) of M_k' are kept the same as $a_1, a_2,\ldots,a_m$. However, the original precedence constraint for M_k is usually not satisfied for M_k'. The precedence constraint is very important for a schema tree. In order to obtain a proper precedence constraint after merging the constraint matrices, the following definition is introduced.

Definition 5. [Score vector of attribute sequences for schema trees]

Suppose that there are n query interfaces which correspond to n schema trees, in which there are m different attributes in total. For any given sequence $a_1, a_2,\ldots, a_m$ of these m attributes, suppose the schema tree T_k of the kth query interface has m' attributes whose sequence in T_k is denoted as $Attri_k$. Then for T_k, define a vector $Vec_k = [v_1, v_2,\ldots,v_i,\ldots,v_m]$, called the score vector for the schema tree T_k in sequence $a_1, a_2,\ldots, a_m$, as follows:

$$v_i = \begin{cases} pos(a_i)/m' & \text{if } a_i \in Attri_k \\ 0 & otherwise \end{cases} \tag{4}$$

where $pos(a_i)$ is the position number of a_i from left to right in sequence $Attri_k$, and note that the position number starts from 1. The reason for this Definition 5 is explained in Definition 7.

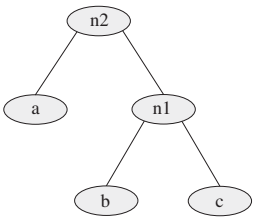
For example, there are three query interfaces which have totally 6 different attributes ($m = 6$). For a given sequence $a_1, a_2,\ldots,a_6$ of these 6 attributes, suppose the second schema tree T_2 has 3 attributes ($m' = 3$) in a sequence a_2, a_1, a_5 ($Attri_2 = \{a_2, a_1, a_5\}$) in T_2, and then the index number of a_1 from left in sequence $Attri_2$ in T_2 is 2, i.e., pos(a_1) = 2, and $m' = 3$, so $v_1 = 2/3$. Similarly, we can get $v_2 = 1/3$, $v_5 = 3/3 = 1$, and the other $v_i = 0$. Thus, $Vec_2 = [2/3, 1/3, 0, 0, 1, 0]$.

Obviously, $1 \le pos(a_i) \le m'$ and $0 \le v_i \le 1$. The smaller the value v_i, the more leftward the attribute a_i located in the original sequence of T_k except for $v_i = 0$.

3.2. Multi-objective optimization model for integrating query interfaces

Before the optimization model is set up, the following observations can be obtained: 1) If the distances between two attributes i and j in all query interfaces are the same, which means all designers for query interfaces think this distance is proper, then the

a) Temporary schema tree T''. b) Temporary constraint matrix M''.



	n2	d
n2	1	2
d	2	1

Fig. 5. The temporary schema tree T'' and its corresponding constraint matrix M'' after removing node n_1 and a which are substituted by its parent node n_2.

distance between them should be kept after integration; 2) If the distances between i and j in most query interfaces are close to a fixed value Dis, then it is very possible that the distance between i and j should also be close to Dis after integration.

Although there are enormous domain-specific query interfaces in the Web, few hierarchical structures of the query interfaces are complicated. Statistics data has shown that the number of levels in the query interfaces (or depth) is generally no more than 5 [3,4,30]. Also, it should be noted that the sequences and group constraints of most attributes in most query interfaces are relatively reasonable because each query interface must be specifically designed. Therefore, the average distance between attributes i and j in all query interfaces should be a good choice as an expected distance, and it can be used as an ideal distance in the united query interface. Note that when $i=j$, this distance becomes the average depth of the attribute i. Similarly, the average sequence position can be used as an ideal sequence position of any attribute.

For a precise description, we introduce the following definitions of the average constraint matrix and average score vector.

Definition 6. [The average constraint matrix/score vector]

For given n order m extended constraint matrices M_1', M_2', ..., and M_n', their average constraint matrix denoted by $avgM$ is defined as $\sum_{k=1}^{n} M_k'[i,j]/n'$, where n' is the number of the non-zero elements of M_1', M_2', ..., and M_n' on row i and column j. For given m-dimensional score vectors Vec_1, Vec_2, ..., and Vec_n, their average vector denoted by $avgVec$ is defined as $\sum_{k=1}^{n} Vec_k'[i]/n'$.

Definition 7. [Merging operation of the constraint matrices/vectors]

Given n order m extended constraint matrices M_1', M_2', ..., and M_n' (note that the original precedence constraints are lost in these matrices, see more details in Definition 4), a merging operation between M_i' and M_j', denoted by $M_i' \oplus M_j'$, is a kind of specific computation on them resulting in an m-order matrix, where the resultant matrix should satisfy the two types of constraints, hierarchical constraints and group constraints, of M_i' and M_j' as much as possible. Similarly, $M_1' \oplus M_2' \oplus \ldots \oplus M_n'$ is a sequence of this kind of computations on these n matrices resulting in an m-order matrix, where the resultant matrix should satisfy the two types of constraints of all M_1', M_2',...,and M_n' as much as possible. Similarly, we can define the merging operation $Vec_1 \oplus Vec_2 \oplus, \ldots, \oplus Vec_n$ on score vectors Vec_1, Vec_2, ..., and Vec_n, where the resultant vector Vec should satisfy the precedence constraints of all Vec_1, Vec_2, ..., and Vec_n. as much as possible.

According to Definition 6 and the above analysis, it can be seen that the average constraint matrix $avgM$ can be seen as an expected constraint matrix, and that the average vector $avgVec$ can be seen as an expected score vector. Our objective is to look for a specific schema tree T whose corresponding constraint matrix M and score vector Vec satisfy the following conditions: 1) They should be as close to $avgM$ and $avgVec$ as possible, respectively. 2) M is generated by merging operation on M_1', M_2',...,and M_n', and Vec is generated by merging operation on $Vec_1 \oplus Vec_2 \oplus, \ldots, \oplus Vec_n$. 3). M should satisfy the hierarchical constrains and group constrains, and Vec should satisfy the precedence constraint. Specifically speaking, we want to find a schema tree with m leaves being all (or, alternatively, the most significant) attributes of n query interfaces such that 1) The error between M and $avgM$ is minimized, and the error between Vec and $avgVec$ is also minimized; 2) $M = M'_1 \oplus M'_2 \oplus \ldots \oplus M'_n$ and $Vec = Vec_1 \oplus Vec_2 \oplus, \ldots, \oplus Vec_n$; 3) M should satisfy the hierarchical constraint and group constrains, denoted by $M \in \{H,G\}$, and Vec should satisfy the precedence constraint, denoted by $Vec \in \{P\}$. The error between M and $avgM$ and the error between Vec and $avgVec$ are defined as follows:

$$MSE(M, avgM) = \sqrt{\frac{\sum_i \sum_j (M[i,j] - avgM[i,j])^2}{m^2}} \quad 1 \le i,j \le m \tag{5}$$

$$MSE(Vec, avgVec) = \sqrt{\frac{\sum_i (Vec[i] - avgVec[i])^2}{m}} \quad 1 \le i \le m. \tag{6}$$

Thus, we can set up a multi-objective optimization problem model for the query interfaces integration problem as follows:

$$\begin{cases} \min & f_1 = MSE(M, avgM) \\ \min & f_2 = MSE(Vec, avgVec) \\ s.t. & M = M_1' \oplus M_2' \oplus \cdots \oplus M_n' \\ & Vec = Vec_1 \oplus Vec_2 \oplus \cdots \oplus Vec_n \\ & M \in \{H, G\} \\ & Vec \in \{P\} \end{cases}. \tag{7}$$

From above Eqs. (5) and (6), we can see that the first objective f_1 requires M should be as close to $avgM$ in mean as possible, and the second objective f_2 requires the order of the nodes of the unified schema tree should be as close to $avgVec$ in mean as possible.

Note that when we look for an ideal integrated interface which minimized the first objective and satisfied the hierarchical constraint and group constraint, e.g., leaf nodes a and b satisfy the group constraint (nodes a and b have a common parent node with $a \prec b$ or $b \prec a$, but not $a \prec \ldots \prec b$ or $b \prec \ldots \prec a$), however, it is very possible that optimization of the second objective requires nodes a and b satisfy the precedence constraint such as $a \prec \ldots \prec b$ or $b \prec \ldots \prec a$. Thus, these two objectives are often conflicting.

For multi-objective optimization problems, it is well known that we shall get a set of optimal solutions called Pareto optimal solutions if we use a usual multi-objective optimization method to solve it. This will result in a new problem on how to choose a proper solution among this set of obtained Pareto optimal solutions, and then need us to make the further decision. However, for the problem of domain-specific query interfaces integrating, we only need one solution. For this purpose and to solve the problem conveniently and obtain an approximate solution to it, in this paper, we construct a new algorithm to find the approximate optimal solution to model (7).

4. Multi-objective optimization integration algorithm of query interfaces based on the attribute constraints

Due to the exponential increase of the domain-specific online WDBs in quantity, the optimization problem above is NP-complete [7,12]. Thus, the solution algorithm must be scalable, suitable for a large-scale integration of the query interfaces. To achieve this purpose, it is necessary to reduce the search space as much as possible. Because the *avgM* /*avgVec* is the expected constraint matrix/score vector which contains rich constraint information worth exploring thoroughly. By a deep observation, it is seen that the search space can be reduced, that is, the operations (computations) on M_1, M_2, …, and M_n /Vec_1, Vec_2, …, and Vec_n can be pruned to the operations (computations) on *avgM* /*avgVec*.

The proposed integration algorithm consists of three algorithms *GetUnifiedConstraintMatrix*, *SeqVal* and *ListSort*. To optimize the first objective by satisfying the hierarchical constraints and group constraints, algorithm *GetUnifiedConstraintMatrix* is designed to get such an approximate optimal solution. To optimize the second objective, algorithms *SeqVal* and *ListSort* are designed to get the approximate optimal solution.

4.1. Algorithm GetUnifiedConstraintMatrix

Based on Definition 3 and its properties of the attribute constraint matrix of a query interface in Section 2, it is evident that the objective function f_1 is closely related to hierarchical constraints and group constraints, and to optimize the objective f_1 is to let M be an integer matrix which is closest to *avgM* and satisfies the hierarchical constraints and group constraints, i.e., the optimal M should be an integer matrix satisfying these facts: 1) M is the closest to *avgM*; 2) $M[i,i]$ should satisfy hierarchical constraint as much as possible, i.e., it should also be an integer which is the closest to $avgM[i,i]$ in feasible search space; and 3) $M[i,j]$ $(i \neq j)$ should satisfy the group constraint as much as possible, i.e., it should be also an integer which is closest to $avgM[i,j]$ $(i \neq j)$ in feasible search space.

Intuitively, M should be integer matrix whose elements are the closest integers to those of the *avgM*. An efficient algorithm *GetUnifiedConstraintMatrix* for obtaining such an approximately optimal M is developed which is shown in Fig. 6.

Since $avgM[i,i]$ is actually a real number (maybe not an integer), while $M[i,i]$ (which represents the depth of node i in its schema tree) should be an integer, an easy way to get M is that we can round the values $avgM[i,i]$ reasonably so that the rounded value satisfies the hierarchical constraints. This can be realized as follows: Assuming that the noise of the query interfaces has been removed (which can be done by the proposed method shown in Section 4.5 in detail.), the hierarchical constraint threshold $BOUND_h$ can be taken as 0.5. If the decimal value of the $avgM[i,i]$ is greater than the $BOUND_h$, then let $M[i,i]$ = the integer part of

Algorithm **GetUnifiedConstraintMatrix** (*avgM, Vec*)

Input:

avgM: an average constraint matrix for n query interfaces.

Vec : a vector of m attributes $a_1, a_2, \ldots, a_m$ for n query interfaces.

Output:

M : a merged constraint matrix that approximately minimizes the objective function f_1.

```
1  M=avgM
   //adjust every element value of the M[i,i] based on avgM[i,i], where 1 ≤ i ≤ m
2  M=RoundDiagElements(avgM);
   //adjust the distance between every two elements of the M[i,j] based on avgM[i,j], where 1 ≤ i,j ≤ m, i ≠ j
3  M=RoundGroupElements(M, Vec);
4  return M;
```

Fig. 6. The pseudo code of algorithm *GetUnifiedConstraintMatrix*.

avgM[i,i] + 1, otherwise, let *M[i,i]* = the integer part of *avgM[i,i]*. In this way, the diagonal elements of *M* not only are the closest to those of *avgM*, but also satisfy the hierarchical constraint.

In addition to the hierarchical constraint, *M* should also satisfy the group constraint. In other words, its non-diagonal elements in *M* should be closest to those of *avgM*. Note that the distance between any two different attributes is at least 2 in its schema tree, and the distance between attributes in one group from the same level *Depth* (*Depth* ≥ 2) must be 2 based on the Lemma in Section 2. Generally, for an ideal (expected) integrated schema tree (corresponding to the integrated/unified query interface and *M*), for any two nodes (attributes) in one group in this ideal schema tree, their distance in most schema trees will be 2, and their distance in only a few schema trees will be larger than 2, e.g., 3 and 4, but it is impossible for their distance to be too large due to the noise having been removed.

To efficiently get the non-diagonal elements of *M*, we can set a proper upper bound, denoted by $BOUND_g$, on *avgM[i,j]* for any two nodes *i* and *j* in one group. From the above analysis, if nodes *i* and *j* are in one group in an ideal schema tree corresponding to *M*, for most schema trees, their distance is 2, and larger than 2 for only a few schema trees. Thus, without loss of generality, we can suppose that for 60% (most but close to average number) schema trees, the distance between any two attributes which belong to one group in the schema trees is 2, for 30% schema trees, it is 3, and for 10% schema trees, it is 4. Thus, we can get an upper bound or threshold $BOUND_g = 60\% \times 2 + 30\% \times 3 + 10\% \times 4 = 2.5$.

Therefore, when $2 \le avgM[i,j] \le 2.5$, then it is possible for nodes *i* and *j* with same depth greater than or equal to 2 are possible to belong to one group. Note that if a parent node (not the root node) has only one child node and this child node is a leaf node (an attribute), this parent node is clearly redundant and can be deleted. If a parent node has more than one child nodes and these child nodes are leaf nodes, then these child nodes are brother (or sister) nodes and they should be in one group. Thus, we can first look for the maximum diagonal elements in *M* whose value is denoted by *Depth* and put their row numbers in a set *SET*, and then for each $i \in SET$, find all possible elements *j* in the *SET* such that $avgM[i,j] <= 2.5$ ($i \neq j$). Furthermore, node *i* and all the nodes *j* are put into the same subset (in the same group). Denote all these subsets as g_1–g_k, respectively. For any other node *i* in *SET* (which does not belong to any subset), its parent node is redundant and can be removed, i.e., let *M[i,i]* = *M[i,i]* − 1. For any two nodes *i* and *j* in a same subset, let *M[i,j]* = 2. For any two nodes *i* and *j* in different subsets, let *M[i,j]* = the even number closest to *avgM*[i,j] in [4, 2*Depth*]. For any node *i* in the *SET* and any leaf node *t* whose parent node is the root, let *M[i,t]* = *Depth* + 1. Update *Depth* by minus 1 and repeat the above process until *Depth* = 1.

In this way, the non-diagonal elements of *M* are not only the closest to those of *avgM*, but also satisfy the group constraint. Thus, *M* obtained above is not only closest to *avgM*, but also satisfies both the hierarchical constraint and group constraint.

The experiments show that aforementioned two threshold values, the $BOUND_h$ and $BOUND_g$, are reasonable. Based on these, the diagonal and non-diagonal elements of *M* can be obtained by functions *RoundDiagElements*(*avgM*) and *RoundGroupElements* of algorithm *GetUnifiedConstraintMatrix* shown in Fig. 7, respectively. For an example and detailed description of the algorithm can be seen in Subsection 4.4.

Note that function *RoundGroupElements* employs a bottom-up approach to determine the every element's value in *M* based on *aveM*, i.e., from the maximum value to the minimum value of *M*[*i,i*]. Once the *M* is determined, the initial unified scheme tree *T*′ of the query interfaces can be obtained by invoking the algorithm *MatrixToTree*.

4.2. Attributes' sort algorithms SeqVal and ListSort

Since the initial unified scheme tree *T*′ above does not contain any information on the precedence constraints of the tree nodes, it is necessary to sort the nodes to optimize the objective function f_2 with satisfying the precedence constraint. Algorithms *SeqVal* and *ListSort* are designed for this purpose. For convenience, the schema tree is denoted as a list. For example, the tree shown in Fig. 2 can be denoted as $((a(bc))d)$, where each (xy) represents nodes *x* and *y* having a parent node with *x* is on the left of *y*, while $(z(xy))$ represents node *z* and the parent node of *x* and *y* having a common parent node with *z* is on the left of the parent node of *x* and *y*, etc. For *avgVec*[*i*] of node *i* in the unified schema tree *T*, the smaller the value, the more leftward the node *i*. Thus, the sequence values for all nodes in the integrated schema tree can be recursively defined from bottom to top as follows. If n_d is a leaf node, $Seq[n_d] = avgVec[n_d]$; if n_d is an internal node, $Seq[n_d]$ = the average of sequence values of all its children nodes, and the smaller the $Seq[n_d]$, the more leftward n_d is. In this way, it is easy to develop a recursive algorithm *SeqVal* (Fig. 8) for assigning an evaluation value to each node and a sorting algorithm *ListSort* (Fig. 9) for sorting all nodes according to their evaluation values, respectively, as follows.

4.3. Analysis for time complexity

Given a set of schema trees T_1, T_2, …, and T_n with *m* different leaf nodes in total, based on the proposed integration algorithm, the constructing processes of the unified schema tree *T* are as follows: 1) constructing the constraint matrices/vectors M_1, M_2, …, and M_n/Vec_1, Vec_2, …, and Vec_n based on these schema trees; 2) determining avg*M*/*avgVec*; 3) computing the unified constraint matrix *M* by the algorithm *GetUnifiedConstraintMatrix*; 4) transforming *M* into initial unified schema tree *T*′ by the algorithm *MatrixToTree*; and 5) achieving the final unified schema tree *T* by the algorithms *SeqVal* and *ListSort*.

Since the time complexity of the some processes above is clear, what follows only makes an analysis of the time complexity of some more complex algorithm.

The main operations in algorithms *GetUnifiedConstraintMatrix*/*MatrixToTree* consist of dividing the nodes in the matrix into different groups (clusters) and updating the matrix by these clusters. Assume that the matrix has *m* nodes initially, and that each

Function **RoundDiagElements**(*avgM*)

```
1 M=avgM
2 BOUND_h=0.5        // hierarchy threshold
3 for (i=1; i<= m ; i++)
4   M[i,i] = (int) (M[i,i]+0.999-BOUND_h)
5 return M
```

Function **RoundGroupElements**(*M*,*Vec*)

```
1 BOUND_g=2.5
2 Depth←max{M(i,i)| M(i,i) ≥ 2 , i=1~n}
3 while(Depth>1)
4     look for the maximum diagonal elements in M and put their row numbers in set SET;
5     divide SET into k largest mutually disjointed groups g_1~g_k and make sure the elements belong to the same
      group if and only if the elements value M[i,j]<= BOUND_g,   (i≠j, i<j ≤ m)
6     S_t=Set-(g_1+g_2+…+g_k)   // the elements in S_t  mean that they belong to none of the groups g_1~g_k
7     if S_t≠ Φ
8         for each element in S_t
9             index←take the index value of the element in Vec
10            M[index,index]=Depth-1
11    for two different attributes i and j in {g_1~g_k}
12        if they belong to the same group g_l
13            M[i,j]=2
14        else
15            M[i,j]=the even number closest to avgM[i,j] in [4, 2Depth]
16    for each element i from the groups g_1~g_k
17        M[t,i]=2*Depth+1   // node t is a leaf node and its parent node is the root
18    Depth=Depth-1
19 endwhile
20 return M
```

Fig. 7. The pseudo code of functions *RoundDiagElements* and *RoundGroupElements*.

iteration of the algorithm will at least reduce one node, then the number of comparisons in partitioning k nodes is of the time of $O(k^2)$, and the time complexity for updating a matrix with k nodes is $O(k^2)$, Since the above operations can be executed at most m times, so the time complexity of the two algorithms are $O(1^2+2^2+\ldots+m^2)=O(m^3)$.

The main operations in the algorithm *ListSort* are to evaluate every sub-list's average value, to sort the list with these sub-list values, and then to sort each sub-list recursively. Assume that list *li* has m elements initially, the time complexity for evaluating all sub-lists' *SeqVal* values is $O(m)$, and the number of comparisons must not be greater than the number of comparisons to sort an array of size $mO(mlogm)$, so the time complexity of algorithm *ListSort* is $O(m^2logm)$.

Therefore, the time complexity of the integration algorithms (i.e., the overall time complexity of the above five processes) proposed in this paper is $2[O(nm^2)+O(nm)]+2O(m^3)+O(m^2logm)$.

Studies [4,30] have indicated that the smallest number of the attributes of a query interface schema tree is 1 (i.e., the simple query interface that is similar to the general search engine query interface, such as Google engine query interface), the largest one is 18, and the average one is 6 across all domain-specific query interfaces of WDBs, which means that the value range of the variable m, the total number of the attributes of a query interface schema tree, is generally between 1 and 18. So, for a large-scale integration of query interfaces, i.e., $n \gg m$, the time complexity of the integration algorithm developed in this paper is $O(n)$. When n and m are in the same order of magnitude, the time complexity of the algorithm is $O(m^3)$.

4.4. Case study

In order to clearly explain the proposed multi-objective optimization integration algorithm of query interfaces for the Deep Web based on attribute constraints, we use the following example to demonstrate the procedure of the proposed algorithm.

Fig. 10 shows the four schema trees to be integrated and its corresponding constraint matrices. Here, in the trees, the leaf nodes from different trees having the same name match each other. The zero elements in a constraint matrix show that there are no leaf nodes in the corresponding schema tree.

Algorithm **SeqVal** (*avgVec, li*)

Input:

avgVec: the average precedence constraint vector.

li: the list corresponding to a schema tree T'.

Output:

avgVec[*li*]: the vector with the precedence constraint information of the *li*.

```
1  if (li is a list)
2      sum = 0
3      num = 0
4      for each x in li    // x is the elements in li and it is a sublist
5          sum = sum + SeqVal(aveVec, x)
6          num = num + 1
7      endfor
8      if (num>0)
9          sum = sum / num
10         return sum
11     endif
12 else
13     return avgVec[li];
14 endif
```

Fig. 8. The pseudo code of algorithm *SeqVal* for calculating the sequence value of a list.

According to Definition 6, the calculated *avgM* and *avgVec* are shown in Fig. 11, respectively.

The algorithm *GetUnifiedConstraintMatrix* for finding *M* based on *avgM* is the key to integration algorithm presented in this paper. Fig. 12 shows its four key steps, where the operation in each step is especially marked with a rectangle.

The key components of algorithm *GetUnifiedConstraintMatrix* for finding *M* based on *avgM* are functions *RoundDiagElements* and *RoundGroupElements*, which consist of four key steps. We now describe them for the example as shown in Fig. 12.

Step 1: invoking function *RoundDiagElements* to obtain $M[i,i]$ by rounding the diagonal elements' values of $avgM[i,i]$ based on the threshold $ROUND_h$ ($ROUND_h = 0.5$). The result is shown in Fig. 12(a).

Step 2: invoking function *RoundGroupElements* and execute its steps 2–5 (shown in Fig. 7), in which the goal is to identify the largest disjointed group elements (attributes) in *M* based on the threshold $ROUND_g$ ($ROUND_g = 2.5$) and $avgM[i,j]$ ($i \neq j$). In this study case, it is clear that the set of the elements first is {*b*,*c*,*d*,*e*,*f*,*g*,*h*,*j*,*k*,*l*} with the largest level *Depth* (*Depth* = 2) in $avgM(i \neq j)$. By executing steps 2–5 of function *RoundGroupElements*, the results, which are identified as the largest disjointed groups, $g_1 =$ {*b*,*c*,*d*,*e*}, $g_2 =$ {*f*,*g*,*h*}, and $g_3 =$ {*k*,*l*} can be obtained, as shown in Fig. 12 (b). Note that the element *j* does not belong to any one group above.

Step 3: execute steps 6–10 of function *RoundGroupElements* with the objective of removing a redundant parent node of a leaf node. After step 2 above, if there exists an element, such as the element *c* shown in Fig. 13 (a) or *j* shown in Fig. 12 (c), which

Algorithm **ListSort**(*avgVec*[*li*], *li*)

Input:

avgVec[*li*]: the average precedence constraint vector of the list *li*.

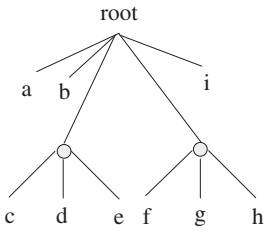
li: a list corresponding to a schema tree T'.

Output:

li: a sorted list corresponding to unified schema tree *T*.

```
1 if (li is a list)
2    sort li according to the value of the avgVec[li]
3        for each x in li    // x is the elements in li and it is a sublist
4            ListSort(avgVec[li], x)
5        endfor
6 endif
```

Fig. 9. The pseudo code of algorithm *ListSort* for sorting a list according to its element values obtained using *SeqVal*.

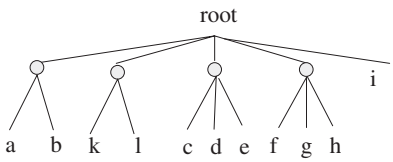


Schema tree T_1

	a	b	c	d	e	f	g	h	i	j	k	l
a	1	2	3	3	3	3	3	3	2	0	0	0
b		1	3	3	3	3	3	3	2	0	0	0
c			2	2	2	4	4	4	3	0	0	0
d				2	2	4	4	4	3	0	0	0
e					2	4	4	4	3	0	0	0
f						2	2	2	3	0	0	0
g							2	2	3	0	0	0
h								2	3	0	0	0
i									1	0	0	0
j										0	0	0
k											0	0
l												0

Vec1 a b c d e f g h i j k l

Corresponding constraint matrix/vector M_1/Vec_1

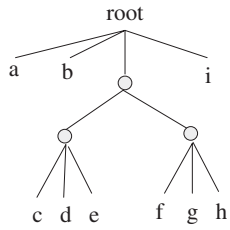


Schema tree T_2

	a	b	c	d	e	f	g	h	i	j	k	l
a	2	2	4	4	4	4	4	4	3	0	4	4
b		2	4	4	4	4	4	4	3	0	4	4
c			2	2	2	4	4	4	3	0	4	4
d				2	2	4	4	4	3	0	4	4
e					2	4	4	4	3	0	4	4
f						2	2	2	3	0	4	4
g							2	2	3	0	4	4
h								2	3	0	4	4
i									1	0	3	3
j										0	0	0
k											2	2
l												2

Vec2 a b k l c d e f g h i j

Corresponding constraint matrix/vector M_2/Vec_2

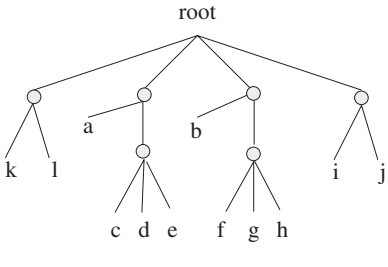


Schema tree T_3

	a	b	c	d	e	f	g	h	i	j	k	l
a	1	2	4	4	4	4	4	4	2	0	0	0
b		1	4	4	4	4	4	4	2	0	0	0
c			3	2	2	4	4	4	4	0	0	0
d				3	2	4	4	4	4	0	0	0
e					3	4	4	4	4	0	0	0
f						3	2	2	4	0	0	0
g							3	2	4	0	0	0
h								3	4	0	0	0
i									1	0	0	0
j										0	0	0
k											0	0
l												0

Vec3 a b c d e f g h i j k l

Corresponding constraint matrix/vector M_3/Vec_3



Schema tree T_4

	a	b	c	d	e	f	g	h	i	j	k	l
a	2	4	3	3	3	5	5	5	4	4	4	4
b		2	5	5	5	3	3	3	4	4	4	4
c			3	2	2	6	6	6	5	5	5	5
d				3	2	6	6	6	5	5	5	5
e					3	6	6	6	5	5	5	5
f						3	2	2	5	5	5	5
g							3	2	5	5	5	5
h								3	5	5	5	5
i									2	2	4	4
j										2	4	4
k											2	2
l												2

Vec4 k l a c d e b f g h i j

Corresponding constraint matrix/vector M_4/Vec_4

Fig. 10. Schema trees to be integrated/merged and their corresponding constraint matrices/vectors.

does not belong to any one group as the only child of its parent node, then there comes the fact that there is a redundant parent node for the element (say element j), so the redundant parent node should be deleted using the operation $M[j,j] = Depth\text{-}1$ (shown in Fig. 12 (c)).

Step 4: execute steps 11–17 of function *RoundGroupElements* to obtain $M[i,j]$ $(i \neq j)$ by revising the elements' values of avg$M[i,j]$ $(i \neq j)$. The fact should be noted that for the elements corresponding to leaf nodes of its schema tree T with the same level $Depth$ ($Depth \geq 2$) in M,1) if these elements belong to a group, the distance between them must be 2, or else 2) the distance between them must be an even number between 4 and 2$Depth$, such as the distance between leaf nodes a and c being 4, that between a and e being 6 ($2Depth = 2 \times 3 = 6$) shown in Fig. 13 (b), and that between a and f being 6 ($< 2Depth = 2 \times 4 = 8$)

a) *avgM*.

	a	b	c	d	e	f	g	h	i	j	k	l
a	1.5	2.5	3.5	3.5	3.5	4	4	4	2.8	1	2	2
b		1.5	4	4	4	3.5	3.5	3.5	2.8	1	2	2
c			2.5	2	2	4.5	4.5	4.5	3.8	1.3	2.3	2.3
d				2.5	2	4.5	4.5	4.5	3.8	1.3	2.3	2.3
e					2.5	4.5	4.5	4.5	3.8	1.3	2.3	2.3
f						2.5	2	2	3.8	1.3	2.3	2.3
g							2.5	2	3.8	1.3	2.3	2.3
h								2.5	3.8	1.3	2.3	2.3
i									1.3	0.5	1.8	1.8
j										2	4	4
k											2	2
l												2

b) *Vec1*~*Vec4* and *avgVec*.

	a	b	c	d	e	f	g	h	i	j	k	l
Vec1	0.11	0.22	0.33	0.44	0.56	0.67	0.78	0.89	1	0	0	0
Vec2	0.09	0.18	0.45	0.55	0.64	0.73	0.82	0.91	1	0	0.27	0.36
Vec3	0.11	0.22	0.33	0.44	0.56	0.67	0.78	0.89	1	0	0	0
Vec4	0.25	0.58	0.33	0.42	0.5	0.67	0.75	0.83	0.92	1	0.08	0.17
avgVec	0.14	0.3	0.36	0.463	0.565	0.685	0.783	0.88	0.98	1	0.175	0.265

Fig. 11. *avgM* and *avgVec*.

shown in Fig. 13 (c), and 3) the distance between the elements and the first level elements must be $Depth+1$, for example, for elements *a* and *g* shown in Fig. 13 (c), their distance is 5 ($Depth+1=4+1=5$). Based on the above facts and the execution of steps 11–17, *M* can be obtained by reasonably revising the element values of avg*M*[*i*,*j*] ($i \neq j$) shown in Fig. 12 (d).

Based on the *avgM*, *avgVec* and the proposed algorithms *MatrixToTree*s, *SeqVal* and *ListSort*, the unified schema tree *T* of the query interfaces' schema trees T_1–T_4, is shown in Fig. 15 after undergoing two iterations (shown in Fig. 14).

In Fig. 14, the first iteration is the operations of algorithm *MatrixToTree* to get temporary constraint matrixes *M*″ based on the initial constraint matrix *M*′ corresponding to an initial unified schema tree *T*′, and the second iteration is the operation of algorithm *ListSort* to sort the precedence order of elements in the *M*″ so as to obtain the constraint matrix *M* corresponding to the final integrated schema tree *T*. Besides, the nodes n_1, n_2, and n_3 are all internal nodes corresponding to the integrated schema tree *T*.

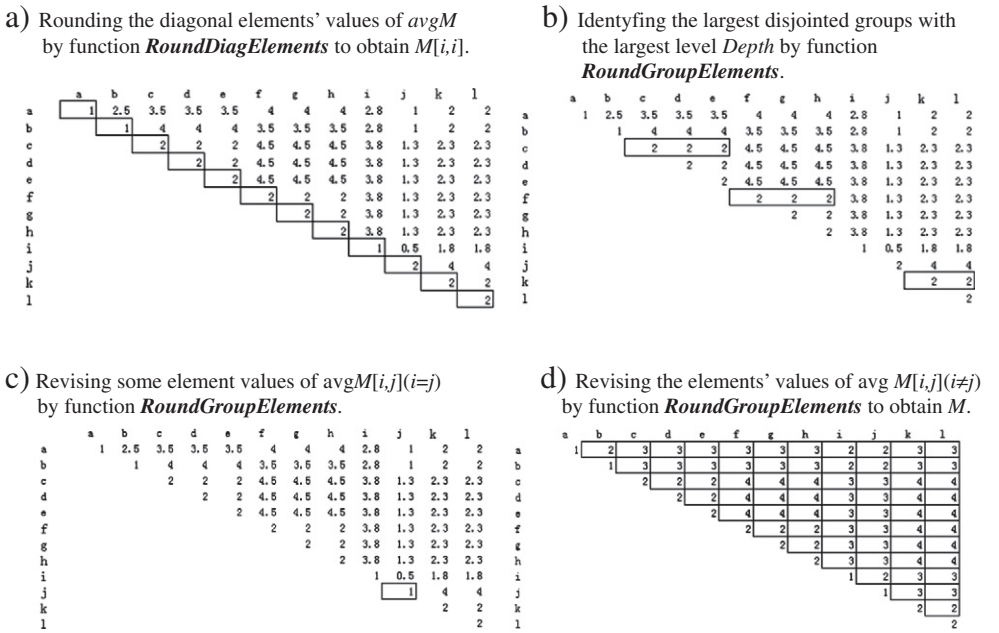


Fig. 12. The four key steps of algorithm *GetUnifiedConstraintMatrix*.

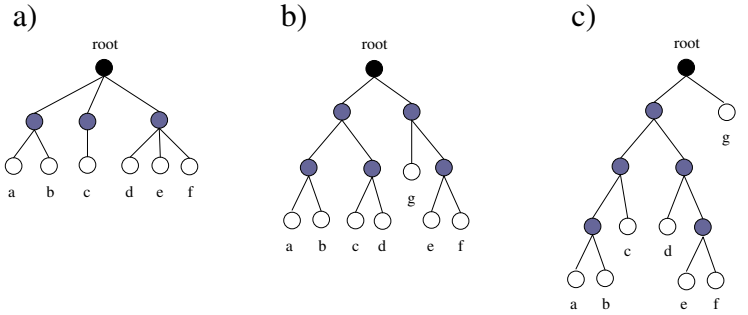


Fig. 13. The Schematic diagrams of distances between the attributes.

4.5. Detecting and filtering out noises

In practice, there may be some attributes from some ill-defined query interfaces (e.g., the attributes of the query interfaces with a flat structure), which are called noises in the context. Since there is a large deviation on the value of the attribute constraints between the ill-defined query interfaces and the query interfaces considered, an error in calculating the average constraint matrix *avgM* and average score vector *avgVec* will be generated. Therefore, when evaluating the average constraint matrix *avgM* and average score vector *avgVec*, the hypothesis test method is adopted in order to detect and remove the noises.

For a value set $(x_1, x_2, x_3, \ldots, x_n)$, its sample mean is $\overline{x}$, and its sample residual is $e_i = x_i - \overline{x}$. If there are no outliers in $(x_1, x_2, x_3, \ldots, x_n)$, then $(x_1, x_2, x_3, \ldots, x_n)$ will obey a normal distribution whose mean is $\overline{x}$, and $(e_1, e_2, \ldots, e_n)$ also obeys a normal distribution whose mean is 0. Therefore, we can test whether there exists a bad value in the sample by hypothesis-testing the normal population variance of $(e_1, e_2, \ldots, e_n)$. If the hypothesis test does not work, it reveals that there is a bad value; if the hypothesis test works, it implies that there is no bad value.

It is an iterative procedure to find out the noise. In each iteration step, the variance of the current sample's residuals is tested by the hypothesis-test method. If the hypothesis is accepted, the iteration is over; if it is rejected, the value which has the maximum deviation from the average is very possible a noise, and then it is removed from the value set, and then the algorithm goes to the next iteration. The process can be represented by an algorithm *avgRNoise* and is shown in Fig. 16.

The TEST procedure used in *avgRNoise* is the hypothesis test, and its specific process is a χ^2 test described below:

$H_0: \sigma^2 \le \sigma_0^2, H_1: \sigma^2 > \sigma_0^2$. The rejection region of the test is $\chi^2 = (n-1)s^2/\sigma_0^2 \ge \chi_\alpha^2(n-1)$. Now let $\sigma_0^2 = 2$ and test level $a = 0.9$.

When H_0 is accepted, TEST procedure returns to "*true*", otherwise, returns to "*false*".

5. Experiments and analysis

5.1. Dataset

The experimental dataset comes from http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/ [28]. The data comes from five domains: airfares, cars, books, jobs and hotel reservation. First, twelve representative query interfaces from two schematically dissimilar and unrelated domains, i.e., airfares domain and books, are selected randomly on each domain for the experiments. Then, in total, 324 query interfaces from aforementioned five domains are selected for the experiments.

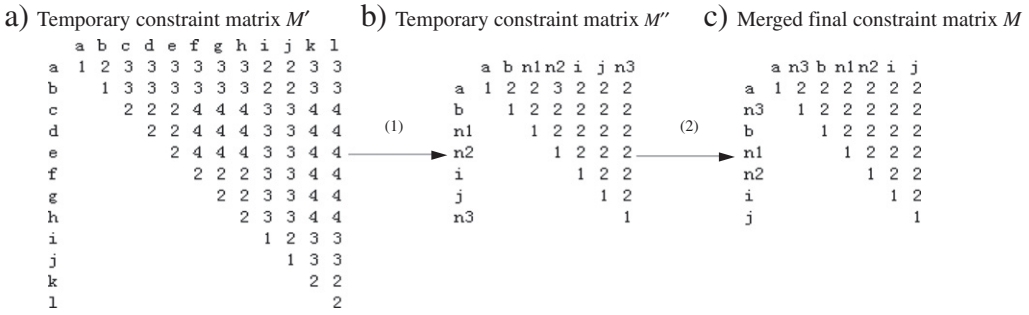
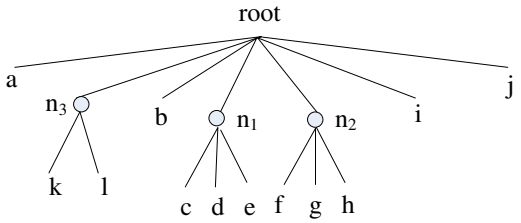


Fig. 14. The two iterations to find integrated schema tree *T* by algorithms *MatrixToTree* and ***ListSort***.

a) The integrated/unified schema tree *T*. b) The integrated/unified constraint matrix *M*.



	a	n3	b	n1	n2	i	j
a	1	2	2	2	2	2	2
n3		1	2	2	2	2	2
b			1	2	2	2	2
n1				1	2	2	2
n2					1	2	2
i						1	2
j							1

Fig. 15. The integrated/unified schema tree *T* and its corresponding constraint matrix *M*.

5.2. Experimental results

Our query interfaces integrating method is based on the fact that the attributes and the three types of constraints among attributes of the query interfaces have been extracted and all the 1:1 mappings have been determined without conflicts by using the method [6].

For simplicity, the schema tree of a query interface is denoted as a list structure defined in Section 4.2. The experimental results of two representative domains, i.e., airfares domain (in which the structure of attributes is a little complex) and books domain (in which the structure of attributes is relatively simple) are shown in the following tables in detail.

Twelve query interfaces from the airfares domain and books domain are selected randomly. The label names of attributes (or attribute names) are displayed in Tables 1 and 3. The schema trees' structures are displayed in Tables 2 and 4.

According to the proposed algorithm in this paper, the final unified query interface of the 12 query interfaces in the airfares domain is ((0, (2, 3), 4), (1, (5, 6), 7), (15, 16), (13, 8, 9), 10), and the final unified query interface of the 12 query interfaces in the books domain is (1, 0, 2, 17, 3, 5, 6, (4), 7, 8, (10, 11, (18, 19), 9, 15, 16), 12, (13, 14)), with the tree structure of the experiment shown in Figs. 17 and 18, respectively, where asterisks represent unnamed internal nodes.

5.3. Discussion on experimental results

Since the proposed integrating algorithm is based on attributes constraints, it is reasonable to evaluate its performance by the rate of the satisfied constraints of the unified query interface among all constraints in each type of the hierarchical, group, and precedence

Algorithm **avgRNoise**(*xSet*)

Input:
xSet: the value set to be processed
Output:
xSet: the processed value set

```
1  while(true)
2      eSet = ∅        // the set of the sample residual
3      update x̄
4      for each x_i in xSet
5          e_i = x_i - x̄
6          add e_i to eSet
7      endfor
8       //TEST(eSet) is a hypothesis testing procedure
9      if(TEST(eSet)==true)
10         return xSet
11     else
12         remove x_i which has the maximum deviation from x̄ from xSet
13     endif
14 endwhile
```

Fig. 16. The pseudo code of algorithm *avgRNoise* for calculating the average value of a value set to remove noises.

Table 1
Label names of attributes in the airfares domain.

No.	Attribute name	No.	Attribute name	No.	Attribute name
0	From	6	Return year	12	Children
1	To	7	Return month	13	Class of service
2	Departure year	8	Return day	14	Airline
3	Departure month	9	Return time	15	Round trip
4	Departure day	10	Senior	16	One way
5	Departure time	11	Adult		

constraints. Since the existing algorithms for query interfaces integrating work on different premises and assumptions, performance comparison in other ways in algorithms make no sense. In order to make a clear and exact description, we begin with some definitions, followed by a comparison and an explanation of the proposed performance measures.

Definition 8. Ordered attributes pair

For any two different attributes i and j in a query interface, $<i, j>$ is called an ordered attributes' pair if i is prior to j in the interface.

For a set of given domain-specific query interfaces, denote the set of their all the ordered attributes pairs by $\boldsymbol{S}$, the subset of $\boldsymbol{S}$ in which the distance between attributes' pair is 2 by $\boldsymbol{G}$, the subset of G in which the distance between ordered attributes pairs is also 2 in the unified query interface by $\boldsymbol{Q}$, the subset of S containing ordered attributes pairs in the unified query interface by $\boldsymbol{T}$, and the subset of S in which the distance between ordered attributes pairs in the query interfaces is not smaller than that in the unified query interface by $\boldsymbol{U}$. Based on these notations, the definition of constraint satisfaction rates can be defined as follows.

Definition 9. Constraint satisfaction rate

$|Q|/|G|$ is called the satisfaction rate of the group constraint; $|T|/|S|$ is called the satisfaction rate of the precedence constraint; and $|U|/|S|$ is called the satisfaction rate of the hierarchical constraint.

The results of the three satisfaction rates in the two domains are shown in Tables 5 and 6, respectively.

After calculations, we get $f_1 = 0.581343591548$ and $f_2 = 0.1102491$ in the airfares domain, while $f_1 = 0.524504711809$ and $f_2 = 0.1316919$ in the books domain by using the proposed integrating algorithm.

Table 5 shows that the integrated interface obtained by the proposed integrating algorithm for airfares domain not only satisfies on average 71.7% group constraints, 88.0% precedence constraints and 75.1% hierarchical constraints, respectively, but also has smaller objective values. The results not only fully illustrate the diversity of domain-specific query interfaces, but also indicate that the proposed algorithm is effective. Moreover, it can be seen from Table 6 that the average constraint satisfaction rates of integrated interface in the books domain are higher than those in airfares domain. This is because the structures of the query interfaces in the airfares domain are generally more complex than those in the books domain, which coincides with the actual situation. Moreover, the values of the objectives f_2 are related to the number of attributes, but the number of attributes of the books domain is greater than that of the airfares domain. So the value f_2 in the airfares domain is slightly higher than that in the books domain, which also coincides with the actual situation.

Furthermore, the query interfaces in the car, jobs, and hotel reservation domains are integrated using the proposed integration algorithm. The average constraint satisfaction rates of the integrated query interface are shown in Fig. 19.

Table 2
Structures of query interfaces in airfares domain.

The numbers of selected query interfaces in TEL-8	Structures of query interfaces
46	(0, 1, (2, 3, 4), (5, 6, 7), 8)
45	((0, 1), (15, 16), (2, 3, 4), (5, 6, 7), 8)
44	(0, 1, (2, 3), (5, 6), 4, 7, 8, 9, 10)
43	(0, 1, ((2, 3, 4), (5, 6, 7)), 8)
42	(0, 1, (2, 3, 4), (5, 6, 7), (8, 13, 9))
41	(((0, 1), ((2, 3), 4)), (((5, 6), 7)), ((15, 16), (8, 9)))
18	((0, 3, 2, 4), (1, 6, 5, 7), ((15, 16), 8, 9))
14	((0, 1), (2, 3, 4), (5, 6, 7), 8)
10	((15, 16), (0, (2, 3, 4)), (1, (5, 6, 7)), (8, 9))
9	((0, (3, 2, 4)), (1, (6, 5, 7)), (15, 16), (8, 10))
8	((0, (2, 3, 4)), (1, (5, 6, 7)), 10, (15, 16))
2	((15, 16), (0, (2, 3, 4)), (1, (5, 6, 7)), 10, (8, 9))

Table 3
Label names of attributes in books domain.

No.	Attribute name	No.	Attribute name	No.	Attribute name
0	Author	7	Country/language	14	Loose
1	Title	8	Binding	15	Singed
2	Keyword	9	First edition	16	Dust jacket
3	ISBN	10	Min price	17	Subject description
4	Item number	11	Max price	18	Publish after
5	Category	12	Response time	19	Publish before
6	Publisher	13	Strict	20	

6. Analysis and comparison with the related work

There is some literature discussing the problem of domain-specific query interfaces integrating [7,11,12,22–27,41]. The works in [7,12] are closely related to our work. Also, the work in [7] models the integration query interfaces as an optimization problem. In this section, we first overview previous researches in the context of the query interfaces integrating for the Deep Web briefly, and then focus on the comparison of our work and the closely related works.

He et al. [23] presented a prototype system MeterQuerier for integrating the domain-specific query interfaces. The MeterQuerier views query interfaces as a visual language, and therefore uses a number of manually pre-defined grammar rules to extract semantically related labels and elements, and then integrates them. However, in the MeterQuerier, the rich semantic/meta information on query interfaces is not considered, with query interfaces modeled as flat schemas, and the efficient and scalable integration query interfaces are not yet discussed. He et al. [11,22,41] developed an automatic and effective query interface extraction and integration tool WISE-Integrator. In their integration strategy, the layout position of each attribute was identified, with local importance of each attribute aggregated, and the more important attributes were arranged ahead of the less important ones. Because group attributes information were ignored, their query interfaces integration method is hardly suitable for the domains having complicated attribute structure, e.g., the airfares domain. Dragut et al. [12] first modeled query interface as an ordered tree of attributes, then provided an efficient pairwise merge algorithm that merged two query interfaces at a time until all query interfaces were merged into a unified query interface, which maintained certain group and hierarchical relationships over the domain-specific query interfaces to be integrated. Since the pairwise merge method was adopted, the algorithm was not holistically merged and is hardly suitable for automatic integration on a large scale of domain-specific query interfaces due to its lower efficiency and local characteristics. Dragut et al. [24] has effectively solved the sub-problem of integration of query interfaces, the reasonable naming the attributes in the unified query interface. Meanwhile, Dragut et al. [25] developed another novel algorithm, which could derive customized integrated interfaces from a domain-specific unified query interface by pruning the nodes of the unified query interface schema tree. Wang [26] proposed a novel method of integration query interfaces based on ontology. However, the details of the method are not available in the published literature. Ma et al. [27] only focused on the discussion of the solution strategy of conflicts in integration query interfaces.

To address a large scale of the problems and the diversity of the query interfaces, Wu et al. [7] presented an optimal integration method of domain-specific query interfaces. Based on modeled hierarchical schema trees of the query interfaces, they first proposed a novel formulation of schema integration and considered it as an optimization problem with the objective of maximally satisfying the constraints given by individual schemas, and then developed a novel approximation algorithm *LMax* which built the unified schema tree of the query interfaces via recursive applications of clustering aggregation. Furthermore, they further extended the algorithm

Table 4
Structures of query interfaces in books domain.

The numbers of selected query interfaces in TEL-8	Structures of query interfaces
1	(0, 1, 2, 3, 4)
4	(2, 1, 0, 5, 3)
5	(0, 1, 6, 2, 3, 7, 8, (9, 15, 16), (10, 11))
6	(0, 1, 2, 3, 8, (10, 11), (9, 15), 12, (13, 14))
7	((1, 0, 17, 2, 6, 4, 3), (8, 7, (10, 11), (18, 19), (9, 15, 16)))
10	(0, 1, 17, 2, 10, 11, (18, 19), 7, 9, 15, 16, 8)
64	((1, 0, 2), (3))
62	((1, 0, 2), (3))
60	(1, 0, 3)
51	(1, 0, 17, 6, (10, 11))
46	((1, 0, 3, 2), ((18, 19)))
36	(3, 2, 0, 6, 17)

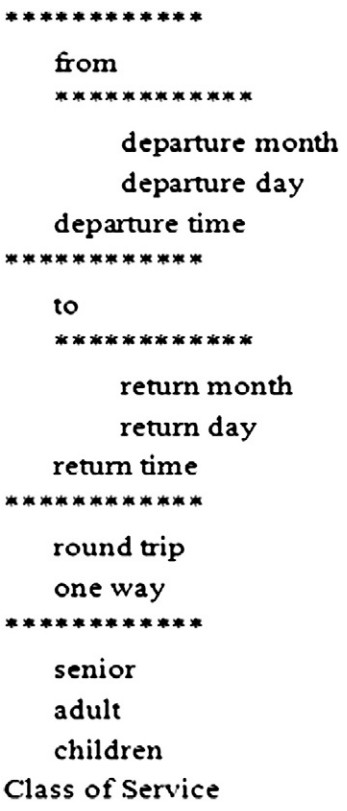


Fig. 17. Unified query interface of airfares domain.

LMax to algorithm *GMax* to handle the irregularities frequently occurring among the interface schema trees of the query interfaces. Extensive evaluation of real-world query interfaces data sets showed the effectiveness of their method.

As an optimization problem and holistic integration of domain-specific query interfaces, several aspects of the difference between our work and the work in Wu et al. [7] are discussed as follows.

6.1. Attribute constraint

Wu et al. [7] first modeled a query interface as a hierarchical ordered tree of the attributes of the query interface, and then identified two types of constraints between the attributes of the query interface, i.e., a structural constraint and a precedence constraint. Moreover, the formal definitions of these two constraints were given. However, the main difference between the proposed modeling and the algorithm from ones in Wu et al. [7] is as follows: First, an additional constraint, called the hierarchical constraint, is explicitly introduced in the proposed model. Second, a novel metric called the constraint matrix which can accurately and quantitatively describe the relationship among attributes (satisfying three types of constraints) is presented and a one to one correspondence between a schema tree and its constraint matrix is proven in this paper. Third, a score vector to the unified schema trees is proposed, which can evaluate its precedence constraint. Fourth, the proposed optimization model is a multi-objective model based on the constraint matrix and a new algorithm is proposed for this model.

6.2. Search space and integration strategy

Wu et al. [7] thought the query interface integration as an optimization problem, but they did not give an explicit model. However, we present a new multi-objective optimization model and a new algorithm for this model. Wu et al. [7] and Dragut et al. [12] adopted clustering aggregation to integrate query interfaces, in which the search space to be explored is $O(nm^2)$, while we only use the simple operations on the constraint matrix and the search space to be explored is $O(m^2)$, where n and m are the total numbers of query interfaces and their corresponding attributes, respectively. For a large number n of the query interfaces, i.e., $n \gg m$, it is obvious that the search space to be explored by our method is much smaller than that by the algorithm in [7]. Thus the proposed algorithm is more efficient.

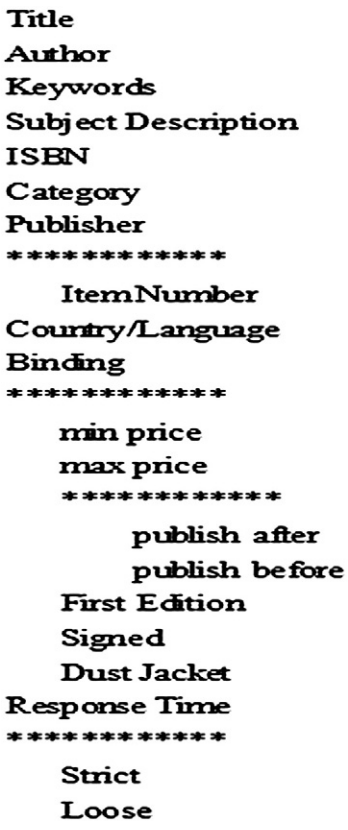


Fig. 18. Unified query interface of books domain.

6.3. Performance of the integration algorithm

Wu et al. [7] adopted the performance measure *PerSC*, which is the percentage of satisfying the strong structural constraints of the integrated/unified interface. According to the above discussion, to exactly describe the unified schema tree, three types of constrains, i.e., hierarchical constraints, group constraints, and precedence constraints, have to be satisfied. Thus, a performance measure which only evaluates the percentage of satisfying the structural constraint is not enough to reflect the performance of an algorithm. Therefore, it is not complete and not enough to use *PerSC* as the performance measure. To measure the performance reasonably and fairly, we propose three metrics: the rate of satisfying hierarchical constraint, the rate of satisfying group constraint and the rate of satisfying precedence constraint, and use them to measure the performance of the proposed algorithm.

Table 5
Constraint satisfaction rates of 12 query interfaces in the airfares domain.

The numbers of selected interfaces in TEL-8	Satisfaction rate of group constraint	Satisfaction rate of precedence constraint	Satisfaction rate of hierarchical constraint
46	0.667	0.917	0.583
45	0.765	0.802	0.736
44	0.600	0.842	0.616
43	0.612	0.857	0.654
42	0.661	0.873	0.696
41	0.672	0.891	0.759
18	0.731	0.905	0.791
14	0.741	0.906	0.788
10	0.763	0.885	0.817
9	0.782	0.896	0.840
8	0.796	0.902	0.855
2	0.810	0.887	0.872
Average satisfaction rate	**0.717**	**0.880**	**0.751**

Table 6
Constraint satisfaction rates of 12 query interfaces in the books domain.

The numbers of selected interfaces in TEL-8	Satisfaction rate of group constraint	Satisfaction rate of precedence constraint	Satisfaction rate of hierarchical constraint
1	0.600	0.900	0.600
4	0.800	0.800	0.800
5	0.911	0.849	0.953
6	0.937	0.908	0.974
7	0.889	0.934	0.949
10	0.726	0.911	0.840
64	0.732	0.913	0.843
62	0.737	0.914	0.844
60	0.742	0.914	0.845
51	0.753	0.918	0.851
46	0.763	0.918	0.857
36	0.777	0.908	0.861
Average satisfaction rate	**0.786**	**0.899**	**0.851**

Their average *PreSC* score ranges from 75.3%–91.2% in structural constraints from five domain query interfaces data sets. Experimental results show that our average performance score on the satisfaction rates of three types of the attribute constraints is 84.5% with the lower optimization objective values.

7. Conclusion and future work

Attempting to address the challenge of the automatic integration of a large number of query interfaces of domain-specific Web Databases, in this paper, through systematic observation and research, we firstly present and prove that there exists a unique attribute constraint matrix for each query interface, which can accurately depict various constraints and the strengths of attributes of the query interface. Then we transform the problem of integrating domain-specific query interfaces into a multi-objective optimization problem model. After that, based on the merging operations on the constraint matrices and the mechanism of automatically detecting and filtering out noises in the query interfaces to be integrated, a novel and efficient algorithm applicable to automatically integrating a large number of the domain-specific query interfaces is developed. In the case of a large number of the domain-specific query interfaces, i.e., $n \gg m$, the time complexity of the proposed algorithm is $O(n)$. Otherwise if n and m are in the same order, the time complexity of our algorithm is $O(m^3)$, where n is the total number of all query interfaces to be integrated, and m is the total number of

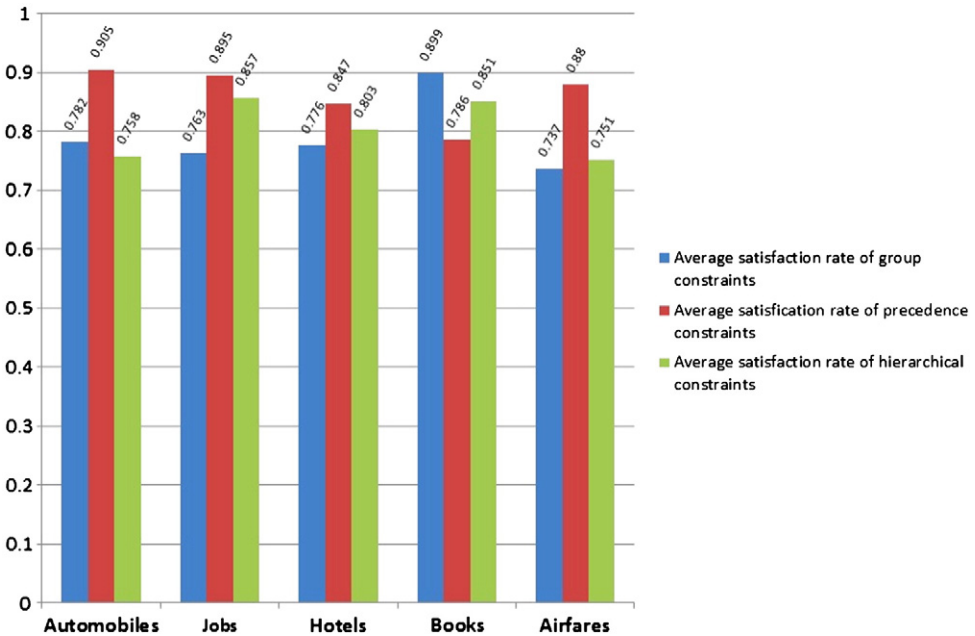


Fig. 19. Average constraint satisfaction rates of various domains.

attributes of all query interfaces, generally, $n \gg m$. Finally, simulation experiments have been carried out with the sample data from real five domains. The results show that the proposed algorithm is effective and efficient, especially for problems with a large number of query interfaces.

The future work will be to improve the integration algorithm so as to further decrease its time complexity, and efficiently tackle the problem of reasonable internal nodes naming by using 1:*m*/*m*:1 and *m*:*n* schema matching results.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (61272119) and Ph.D. Programs Foundation of the Ministry of Education of China (20090203110005).

References

[1] L. Steve, C.L. Giles, Searching the World Wide Web, Science 280 (1998) 98–100.
[2] L. Steve, C.L. Giles, Accessibility of information on the web, Nature 400 (1999) 107–109.
[3] W. Liu, X. Meng, W. Meng, A survey of Deep Web data integration, Chinese Journal of Computers 30 (2007) 1475–1489.
[4] K.C.-C. Chang, B. He, C. Li, M. Patel, Z. Zhang, Structured databases on the web: observations and implications, SIGMOD Record 33 (2004) 61–70.
[5] M.P. Singh, Deep Web structure, IEEE Internet Computing 6 (2002) 4–5.
[6] W. Wu, C. Yu, A. Doan, W. Meng, An interactive clustering-based approach to integrating source query interfaces on the Deep Web, SIGMOD'04 (2004) 13–18.
[7] W. Wu, C. Yu, A. Doan, Merging Interface schemas on the Deep Web via clustering aggregation, ICDM'05 (2005).
[8] R. Dhamankar, Y. Lee, A. Doan, A. Halevy, P. Domingos, iMAP: discovering complex mappings between database schemas, SIGMOD'04 (2004) 13–18.
[9] B. He, K. Chang, J. Han, Discovering complex matchings across web query interfaces: a correlation mining approach, Proceeding KDD'04 Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (2004) 148–157.
[10] W. Wu, A. Doan, C. Yu, WebIQ: learning from the web to match Deep-Web query interfaces, ICDE'06 (2006).
[11] H. He, W. Meng, C. Yu, Z. Wu, WISE-Integrator: a system for extracting and integrating complex web search interfaces of the Deep Web, VLDB'05 (2005) 1314–1317.
[12] E. Dragut, W. Wu, P. Sistla, C. Yu, W. Meng, Merging source query interfaces on web databases, ICDE'06 (2006) 1–10.
[13] S. Melnik, P. Bernstein, A. Halevy, E. Rahm, Supporting executable mappings in model management, SIGMOD'05 (2005) 167–178.
[14] B. He, K.C.-C. Chang, Automatic complex schema matching across web query interfaces: a correlation mining approach, ACM Transactions on Database Systems 31 (2006) 346–395.
[15] E.C. Dragut, T. Kabisch, C. Yu, U. Leser, A hierarchical approach to model web query interfaces for web source integration, VLDB'09 (2009) 325–336.
[16] E. Dragut, F. Fang, P. Sistla, C. Yu, Stop word and related problems in web interface integration, VLDB'09 (2009).
[17] B. He, K.C.-C. Chang, A holistic paradigm for large scale schema matching, SIGMOD Record 33 (2004) 20–25.
[18] B. He, K.C.-C. Chang, Statistical schema matching across web query interfaces, SIGMOD'03 (2003) 9–12.
[19] J. Wang, J.R. Wen, F. Lochovsky, W.Y. Ma, Instance-based schema matching for web databases by domain-specific query probing, VLDB'04 (2004) 408–419.
[20] A. Halevy, A. Rajaraman, J. Ordille, Data integration: the teenage years, VLDB'06, 9–16.
[21] J. Madhavan, S.R. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, A. Halevy, Web-scale data integration: you can only afford to pay as you go, CIRD'07 (2007) 342–350.
[22] H. He, W. Meng, C. Yu, Z. Wu, WISE-Integrator: an automatic integrator of web search interfaces for e-commerce, VLDB'03 (2003).
[23] B. He, Z. Zhang, K. Chang, Knocking the door to the Deep Web: integration of web query interfaces, SIGMOD'04 (2004).
[24] E.C. Dragut, C. Yu, W. Meng, Meaningful labeling of integrated query interfaces, VLDB'06, 679–690.
[25] E.C. Dragut, F. Fang, C. Yu, W. Meng, Deriving customized integrated web query interfaces, IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Workshops, 2009, pp. 685–688.
[26] Y. Wang, T. Peng, Automatic integration of Deep Web query interfaces based on ontology, Fourth International Conference on Computer Sciences and Convergence Information Technology, 2009, pp. 1654–1659.
[27] A. Ma, D. Sun, B. Zhang, K. Gao, Y. Zhang, The classification and solution strategy of conflicts in Deep Web query interface integration, WiCOM'08 (2008) 1–4.
[28] http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/.
[29] Z. Zhang, B. He, K.C.-C. Chang, Understanding web query interfaces: best-effort parsing with hidden syntax, SIGMOD'04 (2004) 107–118.
[30] B. He, M. Patel, Z. Zhang, K.C.-C. Chang, Accessing the Deep Web: a survey, Communications of the ACM 50 (5) (2007) 94–101.
[31] W. Wu, A. Doan, C.T. Yu, W. Meng, Modeling and extracting Deep-Web query interfaces, Advances in Information and Intelligent Systems 251 (2009) 65–90.
[32] W. Su, J. Wang, F. Lochovsky, Holistic schema matching for web query interfaces, advances in database technology (EDBT'06), Lecture Notes in Computer Science 3896 (2006) 77–94.
[33] B. He, K.C.-C. Chang, J. Han, Mining complex matchings across web query interfaces, in: Proceedings of the 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'04) (2004) 3–10.
[34] E. Rahm, P.A. Bernstein, A survey of approaches to automatic schema matching, The VLDB Journal 10 (2001) 334–350.
[35] R.A. Pottinger, P.A. Bernstein, Merging models based on given correspondences, VLDB'03 (2003).
[36] B.R. El-Gamil, W. Winiwarter, B. Božjć, H. Wahl, Deep Web integrated systems: current achievements and open issues, Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS'11), 2011, pp. 447–450.
[37] R. Khare, Y. An, I.-Y. Song, Understanding Deep Web search interfaces: a survey, SIGMOD Record 39 (2010) 33–40.
[38] T. Kabisch, E.C. Dragut, C. Yu, U. Leser, Deep Web integration with VisQI, VLDB'2010 (2010) 1613–1616.
[39] S. Kumar, V.K. Sharma, Architecture of Deep Web: surfacing hidden value, International Journal of Computer Information Systems 3 (2011) 5–9.
[40] F. Yuan, L. Han, Y. Wei, A Deep Web interface integration approach based on keyword matching and similarity computing, Journal of Computational Information Systems 6 (2009) 1569–1576.
[41] H. He, W. Meng, C. Yu, Z. Wu, Automatic integration of web search interfaces with WISE-Integrator, The VLDB Journal 13 (2004) 256–273.
[42] T.M. Ghanem, W.G. Aref, Databases deepen the web, IEEE Computer 37 (2004) 116–117.
[43] M.P. Zillman, Deep Web Research 2012, http://DeepWeb.us/ 2012.
[44] K.M. Bergman, White paper: The Deep Web: surfacing hidden value, Journal of Electronic Publishing 7 (2001). (http://dx.doi.org/10.3998/3336451.0007.104).
[45] W. Wu, A. Doan, C. Yu, WebIQ: learning from the web to match Deep-Web query interfaces, CIDE'06 (2006).
[46] H. Tan, P. Ghodous, J. Montiel, On-line web database integration, in: Proceedings of the International Conference on Management of Emergent Digital EcoSystems (MEDES '10), ACM, New York, NY, USA, (2010), 240–245.
[47] J.F. Terwilliger, L.M.L. Delcambre, J. Logan, Querying through a user interface, Data & Knowledge Engineering 63 (2007) 774–794.
[48] A. Bozzon, M. Brambilla, S. Ceri, P. Fraternali, Liquid query: multi-domain exploratory search on the web, in: Proceedings of the 19th international conference on World wide web (WWW'10). ACM, New York, NY, USA, (2010), 161–170.

Yanni Li is an Associate Professor at School of Software, and a Ph.D. student in Computer Applications at School of Computer Science and Technology, Xidian University, China. Her research interests include Web Data Mining, Web Database Integration, and Object-Oriented Technology.

Yuping Wang is a Professor and Ph.D. supervisor at School of Computer Science and Technology, Xidian University, China. He received his Ph.D. from the Department of Mathematics, Xi'an Jiaotong University, China in 1993. Currently, his research interests include Optimization Methods, Theory and Application, Evolutionary Computation, Data Mining, and Machine Learning.

Peng Jiang is a Master student at Institute of Computing Technology Chinese Academy of Sciences. His research interests include Software Engineering, High Performance Computing, and Data Mining.

Zhensong Zhang is a Master student at Institute of Software Chinese Academy of Sciences. His research interests include Software Engineering, Human – Computer Interaction, and Data Mining.