Holistic Schema Matching for Web Query Interface

Weifeng Su[1], Jiying Wang[2], and Frederick Lochovsky[1]

[1] Hong Kong University of Science & Technology, Hong Kong
{weifeng,fred}@cs.ust.hk
[2] City University, Hong Kong
wangjy@cityu.edu.hk

Abstract. One significant part of today's Web is Web databases, which can dynamically provide information in response to user queries. To help users submit queries to and collect query results from different Web databases, the query interface matching problem needs to be addressed. To solve this problem, we propose a new complex schema matching approach, Holistic Schema Matching (HSM). By examining the query interfaces of real Web databases, we observe that attribute matchings can be discovered from attribute-occurrence patterns. For example, First Name often appears together with Last Name while it is rarely co-present with Author in the Books domain. Thus, we design a count-based greedy algorithm to identify which attributes are more likely to be matched in the query interfaces. In particular, HSM can identify both *simple matching* and *complex matching*, where the former refers to 1:1 matching between attributes and the latter refers to 1:n or m:n matching between attributes. Our experiments show that HSM can discover both simple and complex matchings accurately and efficiently on real data sets.

1 INTRODUCTION

Today, more and more databases that dynamically generate Web pages in response to user queries are available on the Web. These Web databases compose the *deep Web*, which is estimated to contain a much larger amount of high quality information and to have a faster growth than the static Web [1, 3]. Moreover, data in the deep Web are usually structured, which make them much easier to query using database techniques compared to the unstructured data in the static Web.

While each static Web page has a unique URL by which a user can access the page, most *Web databases* are only accessible through a query interface. Once a user submits a query describing the information that he/she is interested in through the query interface, the Web server will retrieve the corresponding results from the back-end database and return them to the user.

To build a system/tool that helps users locate information in numerous Web databases, the very first task is to understand the query interfaces and help dispatch user queries to suitable fields of those interfaces. The main challenge of such a task is that different databases may use different fields or terms to represent the same concept. For example, to describe the genre of a CD in the MusicRecords domain, Category is used in some databases while Style is used in other databases. In the Books domain, First Name and Last Name are used in some databases while Author is used in others to denote the writer of a book.

In this paper, we specifically focus on the problem of matching across query interfaces of structured Web databases. The query interface matching problem is related to a classic problem in the database literature, *schema matching*, if we define an entry or field in a query interface as an *attribute* and all attributes in the query interface form a *schema* of the interface[3]. Schema matching maps semantically related attributes between pairs of schemas in the same domain. When matching the attributes, we call a 1:1 matching, such as Category with Style, a *simple matching* and a 1:n or m:n matching, such as First Name, Last Name with Author, a *complex matching*. In the latter case, attributes First Name and Last Name form a concept group before they are matched to attribute Author. We call attributes that are in the same concept group *grouping attributes* and attributes that are semantically identical or similar to each other *synonym attributes*. For example, attributes First Name and Last Name are grouping attributes, and First Name with Author or Last Name with Author are synonym attributes.

Discovering grouping attributes and synonym attributes in the query interfaces of relevant Web databases is an indispensable step to dispatch user queries to various Web databases and integrate their results. Considering that millions of databases are available on the Web [3], computer-aided interface schema matching is definitely necessary to avoid tedious and expensive human labor.

Although many solutions have been proposed to solve the schema matching problem, current solutions still suffer from the following limitations:

1. *simple matching*: most schema matching methods to date only focus on discovering simple matchings between schemas [2, 6, 9, 16].
2. *low accuracy on complex matching*: although there are some methods that can identify complex matchings, their accuracy is practically unsatisfactory [5, 12].
3. *time consuming*: some methods employ machine-learning techniques that need a lot of training time and some have time complexity exponential to the number of attributes [8, 10].
4. *domain knowledge required*: some methods require domain knowledge, instance data or user interactions before or during the matching process [2, 5, 8, 14, 16, 17].

In this paper, we propose a new interface schema matching approach, ***H**olistic **S**chema **M**atching* (HSM), to find matching attributes across a set of Web database schemas of the same domain. HSM takes advantage of the term occurrence pattern within a domain and can discover both simple and complex matchings efficiently without any domain knowledge.

The rest of the paper is organized as follows. Section 2 reviews related work and compares our approach to previous approaches. In section 3, we introduce our observations on Web database query interfaces and give an example that motivates our approach. Section 4, the main section of the paper, presents the holistic schema matching approach HSM. Our experiments on two datasets and the results are reported in section 5. Section 6 concludes the paper and discusses several further open research issues.

2 RELATED WORK

Being an important step for data integration, schema matching has attracted much attention [2, 5–10, 12, 14, 16, 17]. However, most previous work either focuses on discover-

[3] The terms “schema” and “interface” will be used in this paper interchangeably.

ing simple matchings only or has un-satisfactory performance on discovering complex matchings. This is because complex matching discovery is fundamentally harder than simple matching discovery. While the number of simple matching candidates between two schemas is bounded by the product of the sizes of the two schemas, the number of complex matching candidates is exponential with respect to the size of the two schemas.

As a result, the performance of some existing complex matching discovery algorithms is not satisfactory. [5] tries to convert the problem of matching discovery into the problem of *searching* in the space of possible matches. [12] views the input schemas as graphs and designs a matching algorithm based on a fixpoint computation using the fact that two nodes are similar when their adjacent nodes are similar. Both approaches can handle simple matchings well (average accuracy around 78% in [5] and 58% in [12]), but their accuracy drops dramatically for complex matchings (around 55% in [5] and negative accuracy in [12]). [17] out performs [5, 12] by utilizing different kinds of information, such as linguistic similarity, type similarity and domain similarity between attributes. However, it also needs user interaction during the matching process to tune system parameters.

Different from most existing approaches, [2] and [16] are notable in that they focus on exploiting instance-level information, such as instance-value overlapping, instead of employing schema-level information, like attribute label/name or schema structures. However, these two approaches can only handle simple matchings. In addition, data instances are very hard to obtain in the Web database environment.

[14, 10] are similar approaches in that they manage to combine multiple algorithms and reuse their matching results. [14] proposes several domain-independent combination methods, such as *max* and *average*, and [10] employs a weighted sum and adapts machine learning techniques to learn the importance of each individual component for a particular domain. Although the approach in [10] is able to learn domain-specific knowledge and statistics, it requires a lot of human efforts to manually identify correct matchings as training data.

In contrast to the above works, our approach is capable of discovering simple and complex matchings at the same time without using any domain knowledge, data instances or user involvement. The HSM approach proposed in this paper can be considered as a single *matcher* that only focuses on exploiting domain-specific attribute occurrence statistics. HSM is specifically designed, and is thus more suitable, for the hidden Web environment where there are a large number of online interfaces to match whose attributes are usually informative in order to be understood by ordinary users. Compared with the above works, HSM is not suitable for a traditional database environment, where there are often only two schemas involved in the matching process and the attribute names could be very non-informative, such as attr1 and attr2, depending on the database designers.

Our HSM approach is very close to DCM developed in [7], which discovers complex matchings holistically using data mining techniques. In fact, HSM and DCM are based on similar observations that frequent attribute co-presence indicates a synonym relationship and rare attribute co-presence indicates a grouping relationship. However, HSM has two major differences (advantages) compared to DCM:

1. *measurement*: DCM defines a H-measure, $H = \frac{f_{01} f_{10}}{f_{+1} f_{1+}}$, to measure the negative correlation between two attributes by which synonym attributes are discovered. Such a measure may give a high score for rare attributes, while HSM's matching

score measure does not have this problem. Suppose there are 50 input schemas, where 25 schemas are $\{A_1, A_3\}$, 24 schemas are $\{A_1, A_4\}$ and the remaining one is $\{A_1, A_2, A_4\}$. In these schemas, A_3 and A_4 are actual synonym attributes appearing a similar number of times and A_2 is a rare and "noisy" attribute that only appears once. According to the negative measure of DCM, the matching score $H_{23} = \frac{1\times25}{1\times25} = 1$, and the matching score $H_{34} = \frac{25\times25}{25\times25}$, also 1. In contrast, HSM measures the matching scores as $X_{23} = 0.96$ and $X_{34} = 12.5$ (see section 4.1). In this extreme case, DCM cannot differentiate frequent attributes from rare attributes, which affects its performance.

2. *matching discovery algorithm*: The time complexity of HSM's matching discovery algorithm is polynomial with respect to the number of attributes, n, while the time complexity of DCM is exponential with respect to n. DCM tries to first identify all possible groups and then discover the matchings between them. To discover grouping attributes, it calculates the positive correlation between all combinations of groups, from size 2 to size n (the worst case). In contrast, HSM only considers the grouping score between every two attributes, and the complex matching is discovered by adding each newly found group member into the corresponding group incrementally. Consequently, HSM discovers the matchings much faster than DCM does.

Our experimental results in section 5.2 show that HSM not only has a higher accuracy than DCM, but is also much more efficient in real Web databases.

3 INTUITION: PARALLEL SCHEMAS

In this section, we first present our observations about interface schemas and interface attributes of Web databases in a domain, on which the HSM approach is based. Then, examples are given to motivate the intuition of HSM.

3.1 Observations

In Web databases, query interfaces are not designed arbitrarily. Web database designers try to design the interfaces to be easily understandable and usable for querying important attributes of the back-end databases. For Web databases in the same domain that are about a specific kind of product or a specific topic, their query interfaces usually share many characteristics:

1. Terms describing or labeling attributes are usually unambiguous in a domain although they may have more than one meaning in an ordinary, comprehensive dictionary. For example, the word title has ten meanings as a noun and two meanings as a verb in WordNet [13]. However it always stands for "the name of a book" when it appears in query interfaces of the Books domain. In particular, because we are dealing with query interfaces, an ambiguous term is usually accompanied by other words to make it unambiguous.
2. According to [8], vocabulary of interfaces in the same domain tends to converge at a relatively small size. It indicates that the same concepts in a domain are usually described by the same set of terms.

3. Synonym attributes are rarely co-present in the same interface. For example, Author and Last Name never appeared together in any query interface that we investigate in the Books domain.
4. Grouping attributes are usually co-present in the same interface to form a "larger" concept. For example, in the Airfares domain, From is usually paired with To to form a concept, which is the same as the concept formed by another frequently co-present attribute pair, Departure city and Arrival city. This phenomenon is recognized as *collocation* in natural language [11] and is very common in daily life.

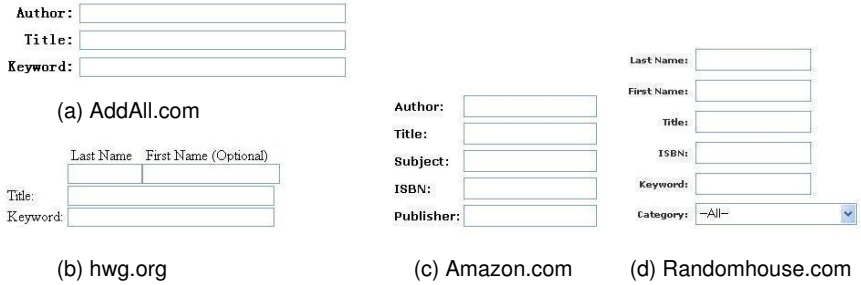

Fig. 1. Examples of query interfaces.

3.2 Motivating Examples

We use the query interfaces shown in Figure 1 to illustrate the main idea of HSM. Let us first consider the schemas in Figure 1(a) and 1(b). The two schemas are semantically equal[4], i.e., any single attribute or set of grouping attributes in one of them semantically corresponds to a single attribute or set of grouping attributes in the other. If we compare the two schemas by putting them in parallel and deleting the attributes that appear in both of them (according to observation 1), we get the matching correspondence between the grouping attributes {First Name, Last Name} and the attribute Author.

Definition 1 *Given two schemas S_1 and S_2, each of which are comprised of a set of attributes, the two schemas form a* ***parallel schema*** *Q, which comprises two attribute sets $\{\{S_1 - S_1 \bigcap S_2\}$ and $\{S_2 - S_1 \bigcap S_2\}\}$.*

AddAll.com	hwg.org
Author	First Name
	Last Name

(a)

Amazon.com	RandomHouse.com
Author	First Name
Subject	Last Name
Publisher	Keyword
	Category

(b)

Table 1. Examples of parallel schemas.

Table 1(a) shows the parallel schema formed by the schemas in Figure 1(a) and 1(b). The complex matching {First Name, Last Name}={Author} is directly available from this parallel schema. However, in most cases, matching is not so easy because

[4] We ignore the word "(Optional)" that appears in Figure 1(b) because it will be discarded during query interface preprocessing [7].

two target schemas may not be semantically equal, such as the schemas in Figure 1(c) and 1(d). After putting these two schemas in parallel and deleting common attributes, the parallel schema in Table 1(b) is obtained. Unfortunately, correct matchings are not directly available from this parallel schema.

To address this problem, we consider any two attributes cross-copresent in a parallel schema to be potential synonym attributes. For example Author with First Name and Author with Last Name in Table1(b) are potential synonym attributes. As a result, if two attributes are potential synonym attributes appearing in many parallel schemas, we may be statistically confident to find the synonym relationship between them (observation 3).

Furthermore, we also notice that First Name and Last Name are always co-present in the same query interface, which indicates that they are very likely to be grouping attributes that form a concept group (observation 4). Suppose we also know that Author with First Name and Author with Last Name are synonym attributes. We can compose an attribute group by First Name and Last Name, with both of the two members matched to Author. That is, {First Name, Last Name}={Author} is discovered as a complex matching.

4 HOLISTIC SCHEMA MATCHING ALGORITHM

We formalize the schema matching problem as the same problem described in [7]. The input is a set of schemas $\mathcal{S} = \{S_1, \ldots, S_u\}$, in which each schema $\mathcal{S}_i$ $(1 \leq i \leq u)$ contains a set of attributes extracted from a query interface and the set of attributes $\mathcal{A} = \cup_{i=1}^{u} S_i = \{A_1, \ldots, A_n\}$ includes all attributes in $\mathcal{S}$. We assume that these schemas come from the same domain. The schema matching problem is to find all matchings $\mathcal{M} = \{M_1, \ldots, M_v\}$ including both simple and complex matchings. A matching M_j $(1 \leq j \leq v)$ is represented as $G_{j1} = G_{j2} = \ldots = G_{jw}$, where G_{jk} $(1 \leq k \leq w)$ is a group of attributes[5] and G_{jk} is a subset of $\mathcal{A}$, i.e., $G_{jk} \subset \mathcal{A}$. Each matching M_j should represent the semantic synonym relationship between two attribute groups G_{jk} and G_{jl} $(l \neq k)$, and each group G_{jk} should represent the grouping relationship between the attributes within it. More specifically, we restrict each attribute to appear no more than one time in $\mathcal{M}$ (observation 1 and 4).

A matching example is {First Name, Last Name} = {Author} in the Books domain, where attributes First Name and Last Name form an attribute group and attribute Author forms another group and the two groups are semantically synonymous. Besides this matching, suppose another matching {Author} = {Writer} is found. According to our restriction, we will not directly include the latter matching in the matching set $\mathcal{M}$. Instead, we may adjust the original matching to {First Name, Last Name} = {Author} = {Writer} or {First Name, Last Name, Writer} = {Author}, depending on whether the relationship found between Writer and {First Name, Last Name} is a grouping or a synonym relationship.

The workflow of the schema matching algorithm is shown in Figure 2. Before the schema matching discovery, two scores, *matching score* and *grouping score*, are calculated between every two attributes. The matching score is used to evaluate the possibility that two attributes are synonym attributes and the grouping score is used to evaluate the possibility that two attributes are in the same group in a matching.

[5] An attribute group can have just one attribute.

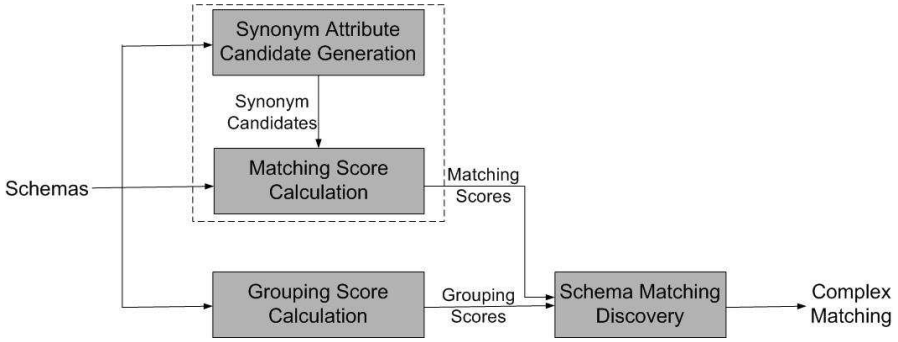

Fig. 2. Holistic Schema Matching Workflow.

The matching score is calculated in two steps. First, *Synonym Attribute Candidate Generation* takes all schemas as input and generates all candidates for synonym attributes based on the observation that synonym attributes rarely co-occur in the same interface schema. Then, *Matching Score Calculation* calculates matching scores between the candidates based on their cross-copresence count (see section 4.1) in the parallel schemas.

Grouping Score Calculation takes all schemas as input and calculates the grouping score between every two attributes based on the observation that grouping attributes frequently co-occur in the same schema.

After calculating the grouping and matching score between every two attributes, we use a greedy algorithm in *Schema Matching Discovery* that iteratively chooses the highest matching score to discover synonym matchings between pairs of attributes. At the same time, the grouping score is used to decide whether two attributes that match to the same set of other attributes belong to the same group. At the end, a matching list is outputted, including both simple and complex matchings. The overall time complexity of HSM is $O(un^2+n^3)$ where n is the number of attributes and u is the number of input schemas. We will explain the time complexity of HSM in detail later in this section.

The rest of this section is organized according to the workflow shown in Figure 2. Subsection 4.1 presents how to calculate the matching score between every two attributes. Subsection 4.2 shows how to calculate the grouping score between every two attributes, and finally subsection 4.3 describes how the matchings can be identified using the grouping and matching scores. In these subsections, the schemas in Table 2 will be used as examples of input schemas.

Table 2. Examples of input schemas.

S_1	S_2	S_3	S_4	S_5
Title	Title	Title	Title	Title
First Name	Author	Author	First Name	Author
Last Name	Subject	Category	Last Name	Category
Category	Publisher			Publisher
Publisher				

4.1 Matching Score Calculation

As discussed above, in HSM the matching scores between two attributes are calculated in two steps: Synonym attribute candidate generation and matching score calculation.

Synonym Attribute Candidate Generation A synonym attribute candidate is a pair of attributes that are possibly synonyms. If there are n attributes in the input schemas, the maximum number of synonym attribute candidates is $C_n^2 = \frac{n(n-1)}{2}$. However, not every two attributes from $\mathcal{A}$ can be actual candidates for synonym attributes. For example in the Books domain, attributes Title and Author should not be considered as synonym attribute candidates, while Author and First Name should. Recall that, in section 3.1, we observed that synonym attributes are rarely co-present in the same schema. In fact, Author and First Name do seldom co-occur in the same interface, while Title and Author appear together very often. This observation can be used to reduce the number of synonym attribute candidates dramatically.

Example 1 *For the four input schemas in Table 2, if we make a strict restriction that any two attributes co-present in the same schema cannot be candidates for synonym attributes, the number of synonym attribute candidates becomes 5 (shown in Table 3), instead of 21 when there is no restriction at all.*

Table 3. Synonym attribute candidates.

1	First Name, Author
2	First Name, Subject
3	Last Name, Author
4	Last Name, Subject
5	Category, Subject

In HSM, we assume that two attributes (A_p, A_q) are synonym attribute candidates if A_p and A_q are co-present in less than $\mathcal{T}_{pq}$ schemas. Intuitively, $\mathcal{T}_{pq}$ should be in proportion to the normalized frequency of A_p and A_q in the input schemas set $\mathcal{S}$. Hence, in our experiments, we set the co-presence threshold of A_p and A_q as

$$\mathcal{T}_{pq} = \frac{\alpha(C_p + C_q)}{u} \tag{1}$$

where α is determined empirically, C_p and C_q are the count of attribute A_p and A_q in $\mathcal{S}$, respectively, and u is the number of input schemas. In out experiments, α is empirically set to be 3. [6]

Suppose there are 50 input schemas and two attributes A_1 and A_2 that occur 20 and 25 times, respectively, then $\mathcal{T}_{12} = 2.7$. This means that A_1 and A_2 should be co-present in no more than two schemas to be synonym attribute candidates.

We use $\mathcal{L} = \{(A_p, A_q), p = 1..n, q = 1..n, p \neq q, C_{pq} < \mathcal{T}_{pq}\}$ to represent the set of synonym attribute candidates, where C_{pq} is the count of the co-occurrences of A_p and A_q in the same schema.

Matching Score Calculation For any two attributes A_p and A_q, a matching score X_{pq} measures the possibility that A_p and A_q are synonym attributes. The bigger the score, the more likely that the two attributes are synonym attributes.

Definition 2 *Given a parallel schema Q, we call A_p and A_q to be **cross-copresent** in Q if $A_p \in S_1 - S_1 \bigcap S_2$ and $A_q \in S_2 - S_1 \bigcap S_2$.*

[6] Experiments have best performance when the $\alpha \in [2, 4]$. We select a middle value of [2,4] here.

If we compare every two schemas, we can get $D_{pq} = (C_p - C_{pq})(C_q - C_{pq})$ parallel schemas in which A_p and A_q are cross-copresent. The bigger D_{pq} is, i.e., the more often A_p and A_q are cross-copresent in a parallel schema, the more likely that A_p and A_q are synonym attributes. However D_{pq} itself is not able to distinguish the scenario as in Example 2:

Example 2 *Suppose there are 50 input schemas, where 15 schemas are* $\{A_1, A_3\}$*, 15 schemas are* $\{A_1, A_4\}$*, 15 schemas are* $\{A_1, A_5\}$ *and the rest 5 are* $\{A_2\}$*. Our intuition is that the matching* $A_3 = A_4 = A_5$ *should be more preferred than matching* $A_1 = A_2$ *because it is highly like that* A_2 *is a noise attribute and occur randomly.* D_{pq} *alone is not able to correctly catch this case because* $D_{12} = D_{34} = D_{35} = D_{45} = 225$*. Meanwhile, we also notice that* $C_1 + C_2 = 50$ *and* $C_3 + C_4 = C_3 + C_5 = C_4 + C_5 = 30$*. Hence if we divide* D_{pq} *by* $C_p + C_q$*, we can reduce the problem caused by noise attributes, such as* A_2 *above.*

Hence, we formulate the matching score between A_p and A_q as:

$$X_{pq} = \begin{cases} 0 & \text{if } (A_p, A_q) \notin \mathcal{L} \\ \frac{(C_p - C_{pq})(C_q - C_{pq})}{(C_p + C_q)} & \textit{otherwise}, \end{cases} \tag{2}$$

Specifically designed for the schema matching problem, this matching score has the following important properties:

1. *null invariance* [15]. For any two attributes, adding more schemas that do not contain the attributes does not affect their matching score. That is, we are more interested in how frequently attributes A_p and A_q are cross co-present in the parallel schemas than how frequently they are co-absent in the parallel schemas.
2. *rareness differentiation*. The matching score between rare attributes and the other attributes is usually low. That is, we consider it is more likely that a rare attribute is cross co-present with other attributes by accident. Example 2 shows the score's penalty over noise attributes.

Example 3 *Matching scores between the attributes from the schemas in Table 2 are shown in Table 4, given the synonym attribute candidates in Table 3.*

Table 4. Matching scores.

	Title	First Name	Last Name	Category	Publisher	Author	Subject
Title		0	0	0	0	0	0
First Name			0	0	0	1.2	0.67
Last Name				0	0	1.2	0.67
Category					0	0	0.75
Publisher						0	0
Author							0
Subject							

In this example, we can see that the matching scores between all the actual synonym attributes are non-zero and high, such as the score between First Name and Author and the score between Category and Subject, which is desirable. The matching scores between some non-synonym attributes are zero, such as the score between Title and Category and the score between Publisher and Author, which is also desirable. However,

the matching scores between some non-synonym attributes are also non-zero yet low, such as the score between First Name and Subject, which is undesirable. To tackle this problem, our matching discovery algorithm is designed to be greedy such that it always considers the matchings with higher scores first when discovering synonym attributes (see section 4.3).

We use $\mathcal{X} = \{X_{pq}, p = 1..n, q = 1..n, p \neq q\}$ to denote the set of matching scores between any two different attributes.

The time complexity for matching score calculation is $O(un^2)$, as there are u schemas to go through and it takes a maximum of $O(n^2)$ time to get the co-occurrence count between any two attributes to generate the synonym candidates and calculate the matching scores.

4.2 Grouping Score Calculation

As mentioned before, a grouping score between two attributes aims to evaluate the possibility that the two attributes are grouping attributes. Recall observation 4 in section 3.1 that grouping attributes are usually co-present in the same interface schema to form a "larger" concept. That is, attributes A_p and A_q are more liable to be grouping attributes if C_{pq} is big. However using C_{pq} only is not sufficient in many cases. Suppose there are 50 input schemas, where 8 schemas are $\{A_1, A_2\}$, 10 schemas are $\{A_1, A_3\}$, 10 schemas are $\{A_3, A_4\}$, and the rest are $\{A_4\}$. In this example, $C_{12} = 8$ and $C_{13} = 10$. Note that A_2 always appears together with A_1 and A_3 does not co-occur with A_1 half of the time, which indicates that A_1 and A_2 are more possible to be a group than A_1 and A_3. Given cases like that, we consider two attributes to be grouping attributes if the less frequent one is usually co-occur with the more frequent one. We propose the following grouping score measure between two attributes A_p and A_q:

$$Y_{pq} = \frac{C_{pq}}{min(C_p, C_q)}. \tag{3}$$

We need to set a grouping score threshold $\mathcal{T}_g$ such that attributes A_p and A_q will be considered as grouping attributes only when $Y_{pq} > \mathcal{T}_g$. Practically, $\mathcal{T}_g$ should be close to 1 as the grouping attributes are expected to co-occur most of the time. In our experiment, $\mathcal{T}_g$ is an empirical parameter and the experimental results show that it has similar performance in a wide range (see section 5.2).

Example 4 *Grouping scores between the attributes from the schemas in Table 2 are shown in Table 5.*

Table 5. Grouping scores between every two different attributes.

	Title	First Name	Last Name	Category	Publisher	Author	Subject
Title		1	1	1	1	1	1
First Name			1	0.5	0.5	0	0
Last Name				0.5	0.5	0	0
Category					0.67	0.67	0
Publisher						0.67	1
Author							1
Subject							

In this example, we can see that the actual grouping attributes First Name and Last Name have a large grouping score, which is desirable. However, it is not very ideal that some non-grouping attributes also have large grouping scores, e.g., Publisher and Subject. This is not a problem in our matching discovery algorithm, which is designed to be matching score centric and always consider the grouping scores together with the matching scores when discovering grouping attributes (see section 4.3).

We use $\mathcal{Y} = \{Y_{pq}, p = 1..n, q = 1..n, p \neq q\}$ to denote the set of grouping scores between any two different attributes. The time complexity of grouping score calculation is $O(un^2)$ as there are u schemas to go through and it takes a maximum of $O(n^2)$ time to go through each schema to obtain the co-occurrence counts for any two attributes.

4.3 Schema Matching Discovery

Algorithm 1 Schema Matching Discovery

Input:
$\mathcal{A} = \{A_i, i = 1...n\}$: the set of attributes from input schemas
$\mathcal{X} = \{X_{pq}, p = 1 \ldots n, q = 1 \ldots n, p \neq q\}$: the set of matching scores between two attributes
$\mathcal{Y} = \{Y_{pq}, p = 1 \ldots n, q = 1 \ldots n, p \neq q\}$: the set of grouping scores between two attributes
$\mathcal{T}_g$: the threshold of grouping score
Output:
$\mathcal{M} = \{M_j, j = 1...v\}$: the set of complex matchings where each matching M_j is represented as $G_{j1} = \ldots = G_{jw}$, and $G_{jk}, k = 1...w$ stands for a group of grouping attributes in $\mathcal{A}$

begin
$\mathcal{M} \leftarrow \emptyset$
while $\mathcal{X} \neq \emptyset$ **do**
    choose the highest matching score X_{pq} in $\mathcal{X}$
    if $X_{pq} = 0$ **then** break;
    end if
    if neither A_p nor A_q appears in $\mathcal{M}$ **then**
        $\mathcal{M} \leftarrow \mathcal{M} + \{\{A_p\} = \{A_q\}\}$
    else if only one of A_p and A_q appears in $\mathcal{M}$ **then**
        /*Suppose A_p appears in M_j and A_q does not appear in $\mathcal{M}$*/
        if For each attribute A_i in M_j, $X_{qi} > 0$ **then**
            $M_j \leftarrow M_j + (= \{A_q\})$
        else if there exists a matching group G_{jk} in M_j such that for any attribute A_l in G_{jk},$Y_{ql} > \mathcal{T}_g$, and for any attribute A_m in other groups $G_{jx}, x \neq k, X_{qm} > 0$ **then**
            $G_{jk} \leftarrow G_{jk} + \{A_q\}$
        end if
    end if
    $\mathcal{X} \leftarrow \mathcal{X} - X_{pq}$
end while
return $\mathcal{M}$
end

With the matching score and grouping score between any two attributes, we propose an iterative matching discovery algorithm, as shown in Algorithm 1. In each iteration,

a greedy selection strategy is used to choose the synonym attribute candidates with the highest matching score (Line 4) until there is no synonym attribute candidate available (Line 5). Suppose X_{pq} is the highest matching score in the current iteration. We will insert its corresponding attributes A_p and A_q into the matching set $\mathcal{M}$ depending on how they appear in $\mathcal{M}$:

1. If neither A_p nor A_q has appeared in $\mathcal{M}$ (Line 7 - 8), $\{A_p\} = \{A_q\}$ will be inserted as a new matching into $\mathcal{M}$.
2. If only one of A_p and A_q has appeared in $\mathcal{M}$ (Line 9 - 16), suppose it is A_p that has appeared in M_j (the j-th matching of $\mathcal{M}$), then A_q will be added into M_j too if:
   - A_q has non-zero matching scores between all existing attributes in M_j. In this case, $\{A_q\}$ is added as a new matching group into M_j (Line 11 - 12).
   - there exists a group G_{jk} in M_j where the grouping score between A_q and any attribute in G_{jk} is larger than the given threshold $\mathcal{T}_g$, and A_q has non-zero matching score between any attribute in the rest of the groups of M_j. In this case, $\{A_q\}$ is added as a member into the group G_{jk} in M_j (Line 13 - 15).
   - If both A_p and A_q have appeared in $\mathcal{M}$, X_{pq} will be ignored because each attribute is not allowed to appear more than one time in $\mathcal{M}$. The reason for this constraint is that if A_p and A_q have been added into $\mathcal{M}$ already, they must have had higher matching scores in a previous iteration.

Finally, we delete X_{pq} from $\mathcal{X}$ (Line 17) at the end of each iteration.

One thing that is not mentioned in the algorithm is how to select the matching score if there is more than one highest score in $\mathcal{X}$. Our approach is to select a score X_{pq} where one of A_p and A_q has appeared in $\mathcal{M}$ but not both. This way of selection makes full use of previously discovered matchings that have higher scores. If there is still more than one score that fits the condition, the selection will be random [7].

Example 4 illustrates the matching discovery iterations using the attributes from the schemas in Table 2.

Example 5 *Before the iteration starts, there is no matching among attributes (Figure 3(a)). In the first iteration,* First Name *with* Author *and* Last Name *with* Author *have the highest matching score from Table 4. As the matching set is empty now, we randomly select one of the above two pairs, say,* First Name *with* Author*. Hence,* {First Name}={Author} *is added to* $\mathcal{M}$ *(Figure 3(b)) and the matching score between* First Name *and* Author *is deleted from* $\mathcal{X}$*. In the second iteration,* Last Name *with* Author *has the highest matching score. Because* Author *has already appeared in* $\mathcal{M}$*,* Last Name *can only be added into the matching in which* Author *appears, i.e.,* {First Name}={Author}*. Suppose the grouping threshold* $\mathcal{T}_g$ *is set to 0.9. We then let* Last Name *form a group with* First Name *as their grouping score is above the threshold (Table 5). Hence, the matching* {First Name}={Author} *is modified to be* {First Name, Last Name}={Author} *in* $\mathcal{M}$ *(Figure 3(c)). After the group is formed, the matching score of* Last Name *with* Author *is deleted from* $\mathcal{X}$*. In the third iteration,* Category *and* Subject *have the highest matching score. Accordingly, the matching* {Category}={Subject} *is added to* $\mathcal{M}$ *(Figure 3(d)) and the matching score between them is deleted from* $\mathcal{X}$*. In the fourth and fifth iterations, no more attributes are added*

[7] Actually tie occurs very seldom in our experiments.

to $\mathcal{M}$ because all attributes associated with the current highest matching score, such as First Name *with* Subject*, have already appeared in $\mathcal{M}$, i.e., they have been matched already. After that, no matching candidates are available and the iteration stops with the final matching results shown in Figure 3(d).*

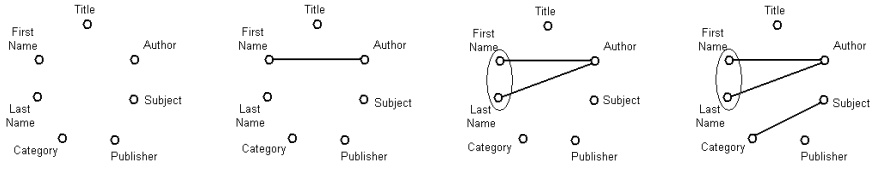

Fig. 3. Matching discovery iterations.

The greediness of this matching discovery algorithm has the benefit of filtering bad matchings in favor of good ones. For instance, in the above example, even though the matching score between First Name and Subject is non-zero, the algorithm will not wrongly match these two attributes because their matching score is lower than the score between First Name and Author, and also lower than the score between Category and Subject.

Another interesting and beneficial characteristic of this algorithm is that it is matching score centric, i.e., the matching score plays a much more important role than the grouping score. In fact, the grouping score is never considered alone without the matching score. For instance in the above example, even though the grouping score between Publisher and Subject is 1, they are not considered by the algorithm as grouping attributes. Recall that a matching {Category}={Subject} is found in the early iterations. In order for Publisher to form a group with Subject, it must have a non-zero matching score with Subject's matching opponent, i.e., Category. Obviously, this condition is not satisfied in the example. Similarly, although Title has high grouping scores with all the other attributes, it forms no groups as its matching score with all the other attributes is zero.

The time complexity of the matching discovery algorithm is $O(n^3)$ because a maximum of n^2 (i.e., the number of scores in $\mathcal{X}$) iterations are needed, and within each iteration a maximum of n comparisons (i.e., the number of attributes in $\mathcal{M}$) are needed.

To conclude, the overall time complexity of HSM is $O(un^2 + n^3)$ since the time complexity of its three steps, matching score calculation, grouping score calculation and schema matching discovery are $O(un^2)$, $O(un^2)$ and $O(n^3)$, respectively.

5 EXPERIMENTS

We choose two datasets, TEL-8 and BAMM, from the UIUC Web integration repository [4], as the testsets for our HSM matching approach. The TEL-8 dataset contains query interface schemas extracted from 447 deep Web sources of eight representative domains: Airfares, Hotels, Car Rentals, Books, Movies, Music Records, Jobs and Automobiles. Each domain contains about 20-70 schemas and each schema contains 3.6-7.2

attributes on average depending on the domain. The BAMM dataset contains query interface schemas extracted from four domains: Automobiles, Books, Movies and Music Records. Each domain has about 50 schemas and each schema contains 3.6-4.7 attributes on average depending on the domain.

In TEL-8 and BAMM, Web databases' query interfaces are manually extracted and their attribute names are preprocessed to remove some irrelevant information, e.g., "search for book titles" is cleaned and simplified to "title". In addition, the data type of each attribute is also recognized in TEL-8 which can be string, integer or datetime. For details of the preprocessing and type recognition, interested readers can refer to [4].

5.1 Metrics

We evaluate the set of matchings automatically discovered by HSM, denoted by $\mathcal{M}_h$, by comparing it with the set of matchings manually collected by a domain expert, denoted by $\mathcal{M}_c$.

To facilitate comparison, we adopt the metric in [7], *target accuracy*, which evaluates how similar $\mathcal{M}_h$ is to $\mathcal{M}_c$. Given a matching set $\mathcal{M}$ and an attribute A_p, a *Closenym set* $Cls(A_p|\mathcal{M})$ is used to refer to all synonym attributes of A_p in $\mathcal{M}$.

Example 6 *For a matching set* $\{\{A_1, A_2\}=\{A_3\}=\{A_4\}\}$*, the closenym set of* A_1 *is* $\{A_3, A_4\}$*, the closenym set of* A_2 *is also* $\{A_3, A_4\}$*, the closenym set of* A_3 *is* $\{A_1, A_2, A_4\}$ *and the closenym set of* A_4 *is* $\{A_1, A_2, A_3\}$*. If two attributes have the same closesym set, they are grouping attributes, such as* A_1 *with* A_2*. If two attribute have each other in their closenym sets, they are synonym attributes, such as* A_1 *with* A_3 *and* A_3 *with* A_4*.*

The target accuracy metric includes *target precision* and *target recall*. For each attribute A_p, the target precision and target recall of its closesym set in $\mathcal{M}_h$ with respect to $\mathcal{M}_c$ are:

$$P_{A_p}(\mathcal{M}_h, \mathcal{M}_c) = \frac{|Cls(A_p|\mathcal{M}_c) \cap Cls(A_p|\mathcal{M}_h)|}{|Cls(A_p|\mathcal{M}_h)|},$$

$$R_{A_p}(\mathcal{M}_h, \mathcal{M}_c) = \frac{|Cls(A_p|\mathcal{M}_c) \cap Cls(A_p|\mathcal{M}_h)|}{|Cls(A_p|\mathcal{M}_c)|}.$$

According to [7], the *target precision* and *target recall* of $\mathcal{M}_h$ (the matching set discovered by a matching approach) with respect to $\mathcal{M}_c$ (the correct matching set) are the weighted average of all the attributes' target precision and target recall (See equ. (4) and (5)). The weight of an attribute A_p is set as $\frac{C_p}{\sum_k C_k}$ in which C_p denotes the count of A_p in $\mathcal{S}$. The reason for calculating the weight in this way is that a frequently used attribute is more likely to be used in a query submitted by a user.

$$P_T(\mathcal{M}_h, \mathcal{M}_c) = \sum_{A_p} \frac{C_p}{\sum_k C_k} P_{A_p}(\mathcal{M}_h, \mathcal{M}_c), \tag{4}$$

$$R_T(\mathcal{M}_h, \mathcal{M}_c) = \sum_{A_p} \frac{C_p}{\sum_k C_k} R_{A_p}(\mathcal{M}_h, \mathcal{M}_c). \tag{5}$$

5.2 Experimental Results

Similar to [7], in our experiment we only consider attributes that occur more than an occurrence-percentage threshold $\mathcal{T}_c$ in the input schema set $\mathcal{S}$, where $\mathcal{T}_c$ is the ratio of the count of an attribute to the total number of input schemas. This is because occurrence patterns of the attributes may not be observable with only a few occurrences. In order to illustrate the influence of such a threshold on the performance of HSM, we run experiments with $\mathcal{T}_c$ set at 20%, 10% and 5%, and show the results below.

Result on the TEL-8 dataset: Table 6 shows the matchings discovered by HSM in the Airfares and CarRentals domains, when $\mathcal{T}_c$ is set at 10%. In this table, the third column indicates whether the matching is correct: Y means fully correct, P means partially correct and N means incorrect. We see that HSM can identify very complex matchings among attributes. We note that destination in Airfares (the fourth row in Table 6) should not form a group by itself to be synonymous to other groups. The reason is that destination co-occurs with different attributes in different schemas, such as depart, origin, leave from to form the same concept, and those attributes are removed because their occurrence-percentages are lower than 10%.

Table 6. Discovered matchings for Airfares and CarRentals when $\mathcal{T}_c = 10\%$.

Domain	*Discovered Matching*	*Correct?*
Airfares	{departure date (datetime), return date (datetime)} = {depart (datetime), return (datetime)}	Y
	{adult (integer), children (integer), infant (integer), senior (integer)} ={passenger (integer)}	Y
	{destination (string)} = {from (string), to (string)} ={arrival city (string), departure city (string)}	P
	{cabin (string)} = {class (string)}	Y
CarRentals	{drop off city (string), pick up city (string)} ={drop off location (string), pick up location (string)}	Y
	{drop off (datetime), pick up (datetime)= { pick up date (datetime), drop off date (datetime), pick up time (datetime), drop off time (datetime)}	Y

Table 7. Target accuracy for TEL-8.

Domain	$\mathcal{T}_c = 20\%$		$\mathcal{T}_c = 10\%$		$\mathcal{T}_c = 5\%$	
	P_T	R_T	P_T	R_T	P_T	R_T
Airfares	1	1	1	.94	.90	.86
Automobiles	1	1	1	1	.76	.88
Books	1	1	1	1	.67	1
CarRentals	1	1	.89	.91	.64	.78
Hotels	1	1	.72	1	.60	.88
Jobs	1	1	1	1	.70	.72
Movies	1	1	1	1	.72	1
MusicRecords	1	1	.74	1	.62	.88
Average	1	1	.92	.98	.70	.88

(a) HSM with $\mathcal{T}_g = 0.9$

Domain	$\mathcal{T}_c = 20\%$		$\mathcal{T}_c = 10\%$		$\mathcal{T}_c = 5\%$	
	P_T	R_T	P_T	R_T	P_T	R_T
Airfares	1	1	1	.71	.56	.51
Automobiles	1	1	.93	1	.67	.78
Books	1	1	1	1	.45	.77
CarRentals	.72	1	.72	.60	.46	.53
Hotels	.86	1	.86	.87	.38	.34
Jobs	1	.86	.78	.87	.36	.46
Movies	1	1	1	1	.48	.65
MusicRecords	1	1	.76	1	.48	.56
Average	.95	.98	.88	.88	.48	.58

(b) DCM

Table 7(a) presents the performance of HSM on TEL-8 when the grouping score threshold $\mathcal{T}_g$ is set to 0.9. As expected, the performance of HSM decreases when we loose the occurrence-percentage threshold $\mathcal{T}_c$ (from 20% to 5%), meaning that more rare attributes are taken into consideration. The phenomenon is because the occurrence pattern of the rare attributes is not obvious with only a few occurrences. Nevertheless, we can see that the performance of HSM is almost always better than the performance of DCM, which was implemented with the optimal parameters reported in [7], especially for a small occurrence percentage threshold such as 5%, as shown in Table 7(b).

We note that the target recall is always higher than the target precision because we do not remove the less likely matchings, which are discovered in later iterations with small matching scores. These less likely matchings will affect the target precision, while they are likely to improve the target recall. One reason that we do not set a threshold to filter lower score matchings is that the threshold is domain dependent. We also consider that it is much easier for a user to check whether a matching is correct than to discover a matching by himself/herself.

Table 8. Target accuracy for BAMM-8

Domain	$\mathcal{T}_c = 20\%$		$\mathcal{T}_c = 10\%$		$\mathcal{T}_c = 5\%$	
	P_T	R_T	P_T	R_T	P_T	R_T
Automobiles	1	1	.56	1	.75	1
Books	1	1	.86	1	.82	1
Movies	1	1	1	1	.90	.86
MusicRecords	1	1	.81	1	.72	1
Average	1	1	.81	1	.80	.97

(a) HSM with $\mathcal{T}_g = 0.9$

Domain	$\mathcal{T}_c = 20\%$		$\mathcal{T}_c = 10\%$		$\mathcal{T}_c = 5\%$	
	P_T	R_T	P_T	R_T	P_T	R_T
Automobiles	1	1	.56	1	.45	1
Books	1	1	.63	1	.47	.78
Movies	1	1	1	1	.45	.53
MusicRecords	1	1	.52	1	.36	.55
Average	1	1	.81	1	.43.3	.72

(b) DCM

Result on the BAMM dataset: The performance of HSM on BAMM is shown in Table 8(a), when the grouping score threshold $\mathcal{T}_g$ is set to 0.9 and the target accuracy of DCM on BAMM is listed in Table 8(b). Again, HSM always outperforms DCM.

We note that the target precision in the Automobiles domain is low when $\mathcal{T}_c = 10\%$. Again, the reason is that we do not remove the matchings with low matching scores, which are less likely to be correct matchings. We also note an exception that, in the Automobiles domain, the precision when $\mathcal{T}_c = 5\%$ is much better than the precision when $\mathcal{T}_c = 10\%$. This is because there are some incorrect matchings identified when $\mathcal{T}_c = 10\%$, while most newly discovered matchings when $\mathcal{T}_c = 5\%$ are correct.

Table 9. Target accuracy of HSM on TEL-8 dataset with different grouping score thresholds when $\mathcal{T}_c = 10\%$.

Domain	$\mathcal{T}_g = .7$		$\mathcal{T}_g = .8$		$\mathcal{T}_g = .9$		$\mathcal{T}_g = .95$	
	P_T	R_T	P_T	R_T	P_T	R_T	P_T	R_T
Airfares	1	.94	1	.94	1	.94	1	.94
Automobiles	1	1	1	1	1	1	1	1
Books	1	1	1	1	1	1	1	1
CarRentals	.69	.71	.75	.81	.89	.91	.86	.88
Hotels	.72	1	.72	1	.72	1	.72	1
Jobs	1	1	1	1	1	1	1	1
Movies	1	1	1	1	1	1	1	1
MusicRecords	.74	1	.74	1	.74	1	.74	1
Average	.89	.96	.90	.97	.92	.98	.92	.98

Table 10. Target accuracy of HSM on BAMM dataset with different grouping score thresholds when $\mathcal{T}_c = 10\%$.

Domain	$\mathcal{T}_g = .7$		$\mathcal{T}_g = .8$		$\mathcal{T}_g = .9$		$\mathcal{T}_g = .95$	
	P_T	R_T	P_T	R_T	P_T	R_T	P_T	R_T
Automobiles	.55	1	.55	1	.55	1	.55	1
Books	.86	1	.86	1	.86	1	.92	1
Movies	1	1	1	1	1	1	1	1
MusicRecords	1	1	1	1	1	1	1	1
Average	.85	1	.85	1	.85	1	.87	1

Influence of grouping score threshold: The performance of HSM with different $\mathcal{T}_g$ on TEL-8 is shown in Table 9. We can see that $\mathcal{T}_g$ actually does not affect the performance of HSM much in a wide range. The target accuracy of HSM is stable with different $\mathcal{T}_g$, except for the target accuracy in domain CarRentals. A similar phenomenon can be observed when we run experiments on BAMM using different $\mathcal{T}_g$, as shown in Table 10. The explanation is as follows:

1. We use a greedy algorithm to always consider high matching scores first and the grouping score plays a minor role in the algorithm. Therefore, the change of grouping score threshold does not make much difference.
2. As we observed, an attribute usually co-occurs with the same set of attributes to form a larger concept. Hence, most grouping attributes have a grouping score equal to 1. This makes the grouping attribute discovery robust to the change of $\mathcal{T}_g$. The reason why the target accuracy in domain CarRentals changes with $\mathcal{T}_g$ is that some attributes in this domain co-occur with different sets of attributes to form the same concept, which makes their grouping scores less than 1 and thus the accuracy is affected by the threshold.

Actual Execution Time: As we have pointed out, HSM discovers matchings in time polynomial to the number of attributes while DCM discovers matchings in time exponential to the number of attributes. In our experiments, both HSM and DCM are implemented in C++ and were run on a PC with an Intel 3.0G CPU and 1G RAM. Table 11 shows the actual execution time accumulated on TEL-8 and BAMM with different $\mathcal{T}_c$. It can be seen that HSM is always order of magnitude faster than DCM. The time needed by DCM grows faster when $\mathcal{T}_c$ is smaller, i.e., when more attributes are considered for matching. It should be noted that DCM takes more than three hours to generate all the matchings when the occurrence-percentage threshold $\mathcal{T}_c = 5\%$.

Table 11. Actual execution time in seconds.

Dataset	$BAMM$			$TEL-8$		
	20%	10%	5%	20%	10%	5%
DCM	0.861	5.171	12.749	2.332	15.813	12624.5
HSM	0.063	0.202	0.297	0.207	0.781	2.313
speedup ratio	13.7	25.6	42.9	11.3	20.2	5458

6 CONCLUSIONS AND FUTURE WORK

In this paper, we present a holistic schema matching approach, HSM, to holistically discover attribute matchings across Web query interfaces. The approach employs several steps, including matching score calculation that measures the possibility of two attributes being synonym attributes, grouping score calculation that evaluates whether two attributes are grouping attributes, and finally a matching discovery algorithm that is greedy and matching score centric. HSM is purely based on the occurrence patterns of attributes and requires neither domain-knowledge nor user interaction. Experimental results show that HSM discovers both simple and complex matchings with very high accuracy in time polynomial to the number of attributes and the number of schemas.

However, we also notice that HSM suffers from some limitations that will be the focus of our future work. In Airfares domain in Table 6, although the matching {from,

to}={arrival city, departure city} has been correctly discovered, HSM is not able to identify the finer matchings {from}={arrival city} and {to}={departure city}. To address this problem, we can consider to employ some auxiliary semantic information (i.e., an ontology) to identify the finer matchings.

We also plan to focus on matching the rare attributes for which HSM's performance is not stable. One promising direction may be to exploit other type of information, such as attribute types, linguistic similarity between attribute names, instance overlapping, and/or schema structures.

Acknowledgment: This research was supported by the Research Grants Council of Hong Kong under grant HKUST6172/04E.

References

1. M. K. Bergman. The deep Web: Surfacing hidden value. http://www.brightplanet.com/technology/deepweb.asp, Dec. 2000.
2. A. Bilke and F. Naumann. Schema matching using duplicates. In *21st Int. Conf. on Data Engineering*, pages 69 – 80, 2005.
3. K. C.-C. Chang, B. He, C. Li, and Z. Zhang. Structured databases on the Web: Observations and implications. Technical Report UIUCDCS-R-2003-2321, CS Department, University of Illinois at Urbana-Champaign, February 2003.
4. K. C.-C. Chang, B. He, C. Li, and Z. Zhang. The UIUC Web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.
5. R. Dhamankar, Y. Lee, A. Doan, A. Halevy, and P. Domingos. imap: Discovering complex semantic matches between database schemas. In *ACM SIGMOD Conference*, pages 383 – 394, 2004.
6. A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM SIGMOD Conference*, pages 509 – 520, 2001.
7. B. He and K. C.-C. Chang. Discovering complex matchings across Web query interfaces: A correlation mining approach. In *ACM SIGKDD Conference*, pages 147 – 158, 2004.
8. B. He, K. C.-C. Chang, and J. Han. Statistical schema matching across Web query interfaces. In *ACM SIGMOD Conference*, pages 217 – 228, 2003.
9. W. Li, C. Clifton, and S. Liu. Database Integration using Neural Network: Implementation and Experience. In *Knowledge and Information Systems,2(1)*, pages 73–96, 2000.
10. J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *21st Int. Conf. on Data Engineering*, pages 57–68, 2005.
11. C. Manning and H. Schutze. *Foundations of Statistical Natural Language Processing*. MIT Press, May, 1999.
12. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *18th Int. Conf. on Data Engineering*, pages 117–128, 2002.
13. G. Miller. *WordNet: An on-line lexical database*. International Journal of Lexicography, 1990.
14. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001.
15. P. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *ACM SIGKDD Conference*, pages 32 – 41, 2002.
16. J. Wang, J. Wen, F. Lochovsky, and W. Ma. Instance-based schema matching for Web databased by domain-specific query probing. In *30-th Int. Conf. Very Large Data Bases*, pages 408–419, 2004.
17. W. Wu, C. Yu, A. Doan, and W. Meng. An interactive clustering-based approach to integrating source query interfaces on the deep Web. In *ACM SIGMOD Conference*, pages 95–106, 2004.